Using VBA to Create Pivot Tables

11

Introducing VBA

Version 5 of Excel introduced a powerful new macro language called Visual Basic for Applications (VBA). Every copy of Excel shipped since 1993 has had a copy of the powerful VBA language hiding behind the worksheets. VBA allows you to perform steps that you normally perform in Excel, but to perform them very very quickly and flawlessly. I've seen a VBA program take a process that would take days each month and turn it into a single button click and a minute of processing time.

Don't be intimidated by VBA. The VBA macro recorder tool will get you 90% of the way to a useful macro and I will get you the rest of the way there using examples in this chapter.

Every example in this chapter is available for download from http://www.mrexcel.com/pivot2007data.html/.

Enabling VBA in Your Copy of Excel

By default, VBA is disabled in Office 2007. Before you can start using VBA, you need to enable macros in the Trust Center. From the Office icon menu, choose Excel Options, Trust Center, Trust Center Settings, Macro Settings.

Choose one of the options below.

- Disable all macros with notification—this setting is equivalent to medium macro security in Excel 2003. When you open a workbook that contains macros, a message will appear alerting that there are macros in the workbook. If you expect macros to be in the workbook, you simply click Options, Enable to allow the macros to run. This is the safest setting, as it forces you to explicitly enable macros in each workbook.

IN THIS CHAPTER

Introducing VBA 231
Learning Tricks of the Trade 234
Understanding Versions 236
Building a Pivot Table in Excel VBA 239
Creating a Report Showing Revenue by Product 246
Handling Additional Annoyances When Creating Your Final Report 250
Addressing Issues with Two or More Data Fields 257
Summarizing Date Fields with Grouping ...263
Using Advanced Pivot Table Techniques 267
Controlling the Sort Order Manually 276
Using Sum, Average, Count, Min, Max, and More 276
Creating Report Percentages 277
Using New Pivot Table Features in Excel 2007 279
Next Steps 289

- Enable all macros (not recommended; potentially dangerous code can run)—this setting is equivalent to low macros security in Excel 2003. Because it could allow rogue macros to run in files that are sent to you by others, Microsoft recommends that you do not use this setting.

> **TIP** If you have previously enabled the Developer tab of the Ribbon, you can use the Macro Security icon to jump quickly to the Trust Center dialog box.

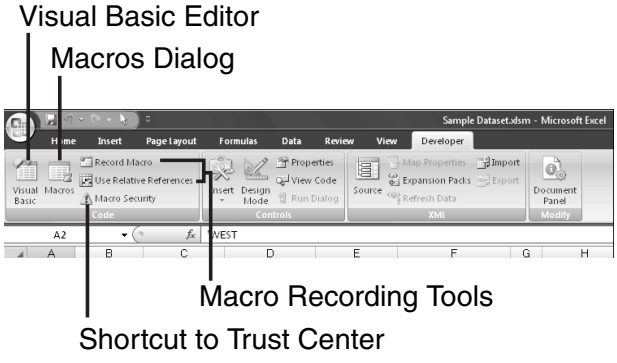


Figure 11.1 Enable the Developer tab to access the VBA tools.

Further, when you save your files, you have to save the files as Excel 2007 macro-enabled workbooks with the `.xlsm` extension.

Enabling the Developer Ribbon

Most of the VBA tools are located on a Developer tab of the Excel 2007 Ribbon. By default, this tab is not displayed. To enable it, from the Office icon menu, select Excel Options, Popular. Then choose Show Developer Tab in the Ribbon.

As shown in Figure 11.1, the Code group on the Developer tab of the Ribbon offers icons for accessing the Visual Basic Editor, Macros dialog box, macro recording tools, and Macro Security setting.

Visual Basic Editor

From Excel, press Alt+F11 or choose Developer, Code, Visual Basic to open the Visual Basic Editor, as shown in Figure 11.2. The three main sections of the VBA Editor are described here. If this is your first time using VBA, some of these items may be disabled. Follow the instructions given in the following list to make sure that each is enabled:

- **Project Explorer**—This pane displays a hierarchical tree of all open workbooks. Expand the tree to see the worksheets and code modules present in the workbook. If the Project Explorer is not visible, enable it by pressing Ctrl+R.

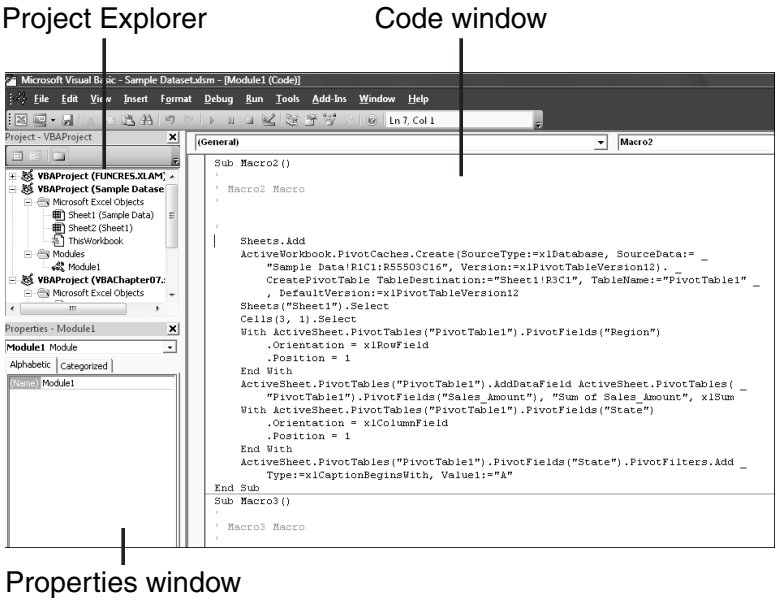


Figure 11.2
The Visual Basic Editor window is lurking behind every copy of Excel shipped since 1993.

- **Properties window**—The Properties window is important when you begin to program user forms. It has some use when you're writing normal code, so enable it by pressing F4.
- **Code window**—This is the area where you write your code. Code is stored in one or more code modules attached to your workbook. To add a code module to a workbook, select Insert, Code Module from the application menu.

Visual Basic Tools

Visual Basic is a powerful development environment. Although this chapter cannot offer a complete course on VBA, if you are new to VBA, you should take advantage of these important tools:

- As you begin to type code, Excel may offer a drop-down with valid choices. This feature, known as AutoComplete, allows you to type code faster and eliminate typing mistakes.
- For assistance on any keyword, put the cursor in the keyword and press F1. You might need your installation CDs because the VBA help file can be excluded from the installation of Office 2007.
- Excel checks each line of code as you finish it. Lines in error appear in red. Comments appear in green. You can add a comment by typing a single apostrophe. Use lots of comments so you can remember what each section of code is doing.

- Despite the aforementioned error checking, Excel may still encounter an error at runtime. If this happens, click the Debug button. The line that caused the error is highlighted in yellow. Hover your mouse cursor over any variable to see the current value of the variable.
- When you are in Debug mode, use the Debug menu to step line by line through code. You can toggle back and forth between Excel and VBA to see the effect of running a line of code on the worksheet.
- Other great debugging tools are breakpoints, the Watch window, the Object Browser, and the Immediate window. Read about these tools in the Excel VBA Help menu.

The Macro Recorder

Excel offers a macro recorder that is about 90% perfect. Unfortunately, the last 10% is frustrating. Code that you record to work with one dataset is hard-coded to work only with that dataset. This behavior might work fine if your transactional database occupies cells A1:K415501 every single day, but if you are pulling in a new invoice register every day, it is unlikely that you will have the same number of rows each day. Given that you might need to work with other data, it would be a lot better if Excel could record selecting cells using the End key. This is one of the shortcomings of the macro recorder.

In reality, Excel pros use the macro recorder to record code but then expect to have to clean up the recorded code.

Understanding Object-Oriented Code

VBA is an object-oriented language. Most lines of VBA code follow the *`Noun.Verb`* syntax. However, in VBA, it is called *`Object.Method`*. Objects can be workbooks, worksheets, cells, or ranges of cells. Methods can be typical Excel actions, such as `.Copy`, `.Paste`, and `.PasteSpecial`.

Many methods allow adverbs—parameters you use to specify how to perform the method. If you see a construct with a colon/equal sign, you know that the macro recorder is describing how the method should work.

You also might see the type of code in which you assign a value to the adjectives of an object. In VBA, adjectives are called *properties*. If you set `ActiveCell.Font.ColorIndex = 3`, you are setting the font color of the active cell to red. Note that when you are dealing with properties, there is only an equal sign, not a colon/equal sign.

Learning Tricks of the Trade

You need to master a few simple techniques to be able to write efficient VBA code. These techniques will help you make the jump to writing effective code.

Writing Code to Handle Any Size Data Range

The macro recorder hard-codes the fact that your data is in a range, such as A1:K415501. Although this hard-coding works for today's dataset, it may not work as you get new datasets. You need to write code that can deal with different size datasets.

The macro recorder uses syntax such as `Range("H12")` to refer to a cell. However, it is more flexible to use `Cells(12, 8)` to refer to the cell in row 12, column 8. Similarly, the macro recorder refers to a rectangular range as `Range("A1:K415501")`. However, it is more flexible to use the `Cells` syntax to refer to the upper-left corner of the range and then use the `Resize()` syntax to refer to the number of rows and columns in the range. The equivalent way to describe the preceding range is `Cells(1, 1).Resize(415501,11)`. This approach is more flexible because you can replace any of the numbers with a variable.

In the Excel user interface, you can use the End key on the keyboard to jump to the end of a range of data. If you move the cell pointer to the final row on the worksheet and press the End key followed by the up-arrow key, the cell pointer jumps to the last row with data. The equivalent of doing this in VBA is to use the following code:

```
Range("A1048576").End(xlUp).Select
```

You don't need to select this cell; you just need to find the row number that contains the last row. The following code locates this row and saves the row number to a variable named `FinalRow`:

```
FinalRow = Range("A1048576").End(xlUp).Row
```

There is nothing magic about the variable name `FinalRow`. You could call this variable `x` or `y`, or even your dog's name. However, because VBA allows you to use meaningful variable names, you should use something such as `FinalRow` to describe the final row.

> **NOTE** Excel 2007 offers 1,048,576 rows and 16,384 columns. Excel 97 through Excel 2003 offered 65,536 rows and 256 columns. To make your code flexible enough to handle any versions of Excel, you can use `Rows.Count` to learn the total number of rows in this version of Excel. The preceding code could then be generalized like so:
>
> ```
> FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
> ```

You also can find the final column in a dataset. If you are relatively sure that the dataset begins in row 1, you can use the End key in combination with the left-arrow key to jump from cell XFD1 to the last column with data. To generalize for the possibility that the code is running in earlier versions of Excel, you can use the following code:

```
FinalCol = Cells(1, Columns.Count).End(xlToLeft).Column
```

End+Down Versus End+Up

You might be tempted to find the final row by starting in cell A1 and using the End key in conjunction with the down-arrow key. Avoid this approach. Data coming from another system is imperfect. If your program will import 500,000 rows from a legacy computer system every day for the next five years, a day will come when someone manages to key a null value into the dataset. This value will cause a blank cell or even a blank row to appear in the middle of your dataset. Using `Range("A1").End(xlDown)` will stop prematurely at the blank cell instead of including all your data. This blank cell will cause that day's report to miss thousands of rows of data, a potential disaster that will call into question the credibility of your report. Take the extra step of starting at the last row in the worksheet to greatly reduce the risk of problems.

Using Super-Variables: Object Variables

In typical programming languages, a variable holds a single value. You might use `x = 4` to assign a value of 4 to the variable `x`.

Think about a single cell in Excel. Many properties describe a cell. A cell might contain a value such as 4, but the cell also has a font size, a font color, a row, a column, possibly a formula, possibly a comment, a list of precedents, and more. It is possible in VBA to create a super-variable that contains all the information about a cell or about any object. A statement to create a typical variable such as `x = Range("A1")` assigns the current value of A1 to the variable `x`.

However, you can use the `Set` keyword to create an object variable:

```
Set x = Range("A1")
```

You've now created a super-variable that contains all the properties of the cell. Instead of having a variable with only one value, you have a variable in which you can access the value of many properties associated with that variable. You can reference `x.Formula` to learn the formula in A1 or `x.Font.ColorIndex` to learn the color of the cell.

Understanding Versions

Pivot tables have been evolving. They were introduced in Excel 5 and perfected in Excel 97. In Excel 2000, pivot table creation in VBA was dramatically altered. Some new parameters were added in Excel 2002. A few new properties such as `PivotFilters` and `TableStyle2` were added in Excel 2007. Therefore, you need to be extremely careful when writing code in Excel 2007 that might be run in Excel 2003 or Excel 2000 or Excel 97.

Just a few simple tweaks make 2003 code run in 2000, but a major overhaul is required to make any code run in Excel 97. Because it has been 10 years since the release of Excel 97 (and because Microsoft has not supported that product for 5+ years), this chapter focuses on using only the pivot cache method introduced in Excel 2000. At the end of the chapter, you briefly learn the PivotTable Wizard method, which is your only option if you need code to run in Excel 97.

New in Excel 2007

Although the basic concept of pivot tables is the same in Excel 2007 as it was in Excel 2003, several new features are available in Excel 2007 pivot tables. The entire Design ribbon is new, including the concepts of subtotals at the top, the report layout options, blank rows, and the new PivotTable styles. Excel 2007 offers better filters than previous versions. It also makes the expand and collapse functionality more apparent by adding buttons to the pivot table grid. Every new feature adds one or more methods or properties to VBA.

If you are hoping to share your pivot table macro with people running prior versions of Excel, you need to avoid these methods. Your best bet is to open an Excel 2003 workbook in compatibility mode and record the macro while the workbook is in compatibility mode. If you are using the macro only in Excel 2007 or later, you can use any of these new features.

Table 11.1 shows the methods that are new in Excel 2007. If you record a macro that uses these methods, you cannot share the macro with someone using Excel 2003 or earlier.

Table 11.1 Methods New in Excel 2007

Method	Description
`ClearAllFilters`	Clears all filters in the pivot table.
`ClearTable`	Removes all fields from the pivot table but keeps the pivot table intact.
`ConvertToFormulas`	Converts a pivot table to cube formulas. This method is valid only for pivot tables based on OLAP data sources.
`DisplayAllMemberPropertiesInTooltip`	Equivalent to Options, Display, Show Properties in ToolTips.
`RowAxisLayout`	Changes the layout for all fields in the row area. Valid values are `xlCompactRow`, `xlTabularRow`, or `xlOutlineRow`.
`SubtotalLocation`	Controls whether subtotals appear at the top or bottom of each group. Valid arguments are `xlAtTop` or `xlAtBottom`.

Table 11.2 lists the properties that are new in Excel 2007. If you record a macro that refers to these properties, you cannot share the macro with someone using Excel 2003 or earlier.

Table 11.2 Properties New in Excel 2007

Property	Description
`ActiveFilters`	Indicates the active filters in the pivot table; this is a read-only property.

continues

Table 11.2 Continued

Property	Description
`AllowMultipleFilters`	Indicates whether a pivot field can have multiple filters applied to it at the same time.
`CompactLayoutColumnHeader`	Specifies the caption that is displayed in the column header of a pivot table when in compact row layout form.
`CompactLayoutRowHeader`	Specifies the caption that is displayed in the row header of a pivot table when in compact row layout form.
`CompactRowIndent`	Indicates the indent increment for pivot items when compact row layout form is turned on.
`DisplayContextTooltips`	Controls whether ToolTips are displayed for pivot table cells.
`DisplayFieldCaptions`	Controls whether filter buttons and pivot field captions for rows and columns are displayed in the grid.
`DisplayMemberPropertyTooltips`	Controls whether to display member properties in ToolTips.
`FieldListSortAscending`	Controls the sort order of fields in the PivotTable Field List. When this property is `True`, the fields are sorted in alphabetical order. When it is set to `False`, the fields are presented in the same sequence as the data source columns.
`InGridDropZones`	Controls whether you can drag and drop fields onto the grid. Changing the pivot table layout also changes this property. Changing this proerty forces the layout back to a table layout.
`LayoutRowDefault`	Specifies the layout settings for pivot fields when they are added to the pivot table for the first time. Valid values are `xlCompactRow`, `xlTabularRow`, or `xlOutlineRow`.
`PivotColumnAxis`	Returns a `PivotAxis` object representing the entire column axis.
`PivotRowAxis`	Returns a `PivotAxis` object representing the entire row axis.
`PrintDrillIndicators`	Specifies whether drill indicators are printed with the pivot table.

Property	Description
`ShowDrillIndicators`	Specifies whether drill indicators are shown in the pivot table.
`ShowTableStyleColumnHeaders`	Controls whether table style 2 should affect the column headers.
`ShowTableStyleColumnStripes`	Controls whether table style 2 should show banded columns.
`ShowTableStyleLastColumn`	Controls whether table style 2 should format the final column.
`ShowTableStyleRowHeaders`	Controls whether table style 2 should affect the row headers.
`ShowTableStyleRowStripes`	Controls whether table style 2 should show banded columns.
`SortUsingCustomLists`	Controls whether custom lists are used for sorting items of fields, both initially and later when applying a sort. Setting this property to `False` can optimize performance for fields with many items and allows you to avoid using custom-list based sorting.
`TableStyle2`	Specifies the pivot table style currently applied to the pivot table. Note that previous versions of Excel offered a weak AutoFormat option. That feature's settings were held in the `TableStyle` property, so Microsoft had to use `TableStyle2` as the property name for the new pivot table styles. The property might have a value such as `PivotStyleLight17`.

Building a Pivot Table in Excel VBA

In this chapter, we do not mean to imply that you use VBA to build pivot tables to give to your clients. Rather, the purpose of this chapter is to remind you that pivot tables can be used as a means to an end; you can use a pivot table to extract a summary of data and then use that summary elsewhere.

> **TIP** The code listings from this chapter are available for download at http://www.MrExcel.com/pivot2007data.html.

CAUTION

Although the Excel user interface has new names for the various sections of a pivot table, VBA code will continue to refer to the old names. Microsoft had to use this choice, otherwise millions of lines of code would stop working in Excel 2007 when they referred to a page field instead of a filter field. While the four sections of a pivot table in the Excel user interface are Report Filter, Column Labels, Row Labels, and Values, VBA continues to use the old terms of Page fields, Column fields, Row fields, and Data fields.

In Excel 2000 and newer, you first build a pivot cache object to describe the input area of the data:

```
Dim WSD As Worksheet
Dim PTCache As PivotCache
Dim PT As PivotTable
Dim PRange As Range
Dim FinalRow As Long
Dim FinalCol As Long
Set WSD = Worksheets("PivotTable")

' Delete any prior pivot tables
For Each PT In WSD.PivotTables
    PT.TableRange2.Clear
Next PT

' Define input area and set up a Pivot Cache
FinalRow = WSD.Cells(Rows.Count, 1).End(xlUp).Row
FinalCol = WSD.Cells(1, Columns.Count).End(xlToLeft).Column
Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:=xlDatabase, _
    SourceData:=PRange)
```


After defining the pivot cache, use the `CreatePivotTable` method to create a blank pivot table based on the defined pivot cache:

```
Set PT = PTCache.CreatePivotTable(TableDestination:=WSD.Cells(2, FinalCol + 2), _
    TableName:="PivotTable1")
```

In the `CreatePivotTable` method, you specify the output location and optionally give the table a name. After running this line of code, you have a strange-looking blank pivot table, like the one shown in Figure 11.3. You now have to use code to drop fields onto the table.

If you choose the Defer Layout Update setting in the user interface to build the pivot table, Excel does not recalculate the pivot table after you drop each field onto the table. By default in VBA, Excel calculates the pivot table as you execute each step of building the table. This could require the pivot table to be executed a half-dozen times before you get to the final result. To speed up your code execution, you can temporarily turn off calculation of the pivot table by using the `ManualUpdate` property:

```
PT.ManualUpdate = True
```

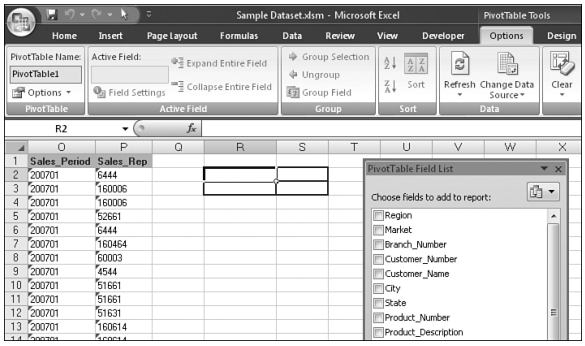


Figure 11.3 Immediately after you use the `CreatePivotTable` method, Excel gives you a four-cell blank pivot table that is not very useful.

You can now run through the steps needed to lay out the pivot table. In the `.AddFields` method, you can specify one or more fields that should be in the row, column, or filter area of the pivot table.

The `RowFields` parameter enables you to define fields that appear in the Row Labels layout area of the PivotTable Field List. The `ColumnFields` parameter corresponds to the Column Labels layout area. The `PageFields` parameter corresponds to the Report Filter layout area.

The following line of code will populate a pivot table with two fields in the row area and one field in the column area.

```
' Set up the row & column fields
PT.AddFields RowFields:=Array("Business Segment", "Product"), _
    ColumnFields:="Region"
```

To add a field such as Revenue to the values area of the table, you change the `Orientation` property of the field to be `xlDataField`.

Getting a Sum Instead of a Count

Excel is smart. When you build a report with revenue, it assumes you want to sum the revenue. But there is a problem. Say that one of the revenue cells is blank. When you build the pivot table, even though 99.9% of fields are numeric, Excel assumes you have alphanumeric data and offers to count this field. This is annoying. It seems to be an anomaly that, on one hand, you are expected to make sure that 100% of your cells have numeric data, but on the other hand, the results of the pivot table are often filled with non-numeric blank cells.

When you build the pivot table in the Excel interface, you should take care in the Values drop zone to notice that the field reads Count of Revenue instead of Sum of Revenue. At that point, the right course of action is to go back and fix the data, but what people usually do is double-click the Count of Revenue button and change it to Sum of Revenue.

In VBA, you should always explicitly define that you are creating a sum of revenue by explicitly setting the `Function` property to `xlSum`:

```
' Set up the data fields
With PT.PivotFields("Revenue")
    .Orientation = xlDataField
    .Function = xlSum
    .Position = 1
End With
```

At this point, you've given VBA all the settings required to correctly generate the pivot table. If you set `ManualUpdate` to `False`, Excel calculates and draws the pivot table. You can immediately thereafter set this back to `True`:

```
' Calc the pivot table
PT.ManualUpdate = False
PT.ManualUpdate = True
```

Your pivot table inherits the table style settings selected as the default on whatever computer happens to run the code. If you would like control over the final format, you can explicitly choose a table style. The following code applies banded rows and a medium table style:

```
' Format the pivot table
PT.ShowTableStyleRowStripes = True
PT.TableStyle2 = "PivotStyleMedium10"
```

At this point, you have a complete pivot table like the one shown in Figure 11.4.

Figure 11.4
Fewer than 50 lines of code create this pivot table in less than a second.

Sum of Reve		Region				
Business	Product	WEST	MIDWEST	NORTH	SOUTH	Grand Total
⊟Housekee	Cleaning & Housekeeping Services	146623.3443	174518.0799	534284.1862	283170.1652	1138595.776
Housekeeping and Organization Total		**146623.3443**	**174518.0799**	**534284.1862**	**283170.1652**	**1138595.776**
⊟Landscapi	Green Plants and Foliage Care	870388.2912	93040.7897	68056.4085	157820.8837	1189306.373
	Landscaping/Grounds Care	365927.5786	190001.943	222555.6228	335674.8795	1114160.024
Landscaping and Area Beautification Total		**1236315.87**	**283042.7327**	**290612.0313**	**493495.7632**	**2303466.397**
⊟Maintenan	Facility Maintenance and Repair	444825.8505	255526.3266	312490.3196	358516.6048	1371359.102
	Fleet Maintenance	521976.0553	448800.6143	337616.4441	1046229.856	2354622.97
	Predictive Maintenance/Preventative Maintenance	655091.1036	478930.1871	456766.1179	472047.328	2062834.737
Maintenance and Repair Total		**1621893.009**	**1183257.128**	**1106872.882**	**1876793.789**	**5788816.808**
Grand Total		**3004832.223**	**1640817.941**	**1931769.099**	**2653459.717**	**9230878.98**

Listing 11.1 shows the complete code used to generate the pivot table.

Listing 11.1 Code to Generate a Pivot Table

```
Sub CreatePivot()
    Dim WSD As Worksheet
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long
    Dim FinalCol As Long
    Set WSD = Worksheets("PivotTable")

    ' Delete any prior pivot tables
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT

    ' Define input area and set up a Pivot Cache
    FinalRow = WSD.Cells(Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Columns.Count). _
        End(xlToLeft).Column
```

```
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:= _
        xlDatabase, SourceData:=PRange)

    ' Create the Pivot Table from the Pivot Cache
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD. _
        Cells(2, FinalCol + 2), TableName:="PivotTable1")

    ' Turn off updating while building the table
    PT.ManualUpdate = True

    ' Set up the row & column fields
    PT.AddFields RowFields:=Array("Business Segment", "Product"), _
        ColumnFields:="Region"

    ' Set up the data fields
    With PT.PivotFields("Revenue")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
    End With

    ' Calc the pivot table
    PT.ManualUpdate = False
    PT.ManualUpdate = True

' Format the pivot table
PT.ShowTableStyleRowStripes = True
PT.TableStyle2 = "PivotStyleMedium10"

End Sub
```


Learning Why You Cannot Move or Change Part of a Pivot Report

Although pivot tables are incredible, they have annoying limitations. You cannot move or change just part of a pivot table. For example, try to run a macro that would delete column X, which contains the Grand Total column of the pivot table. The macro comes to a screeching halt with an error 1004, as shown in Figure 11.5. To get around this limitation, you can change the summary from a pivot table to just values using the PasteSpecial method described below.

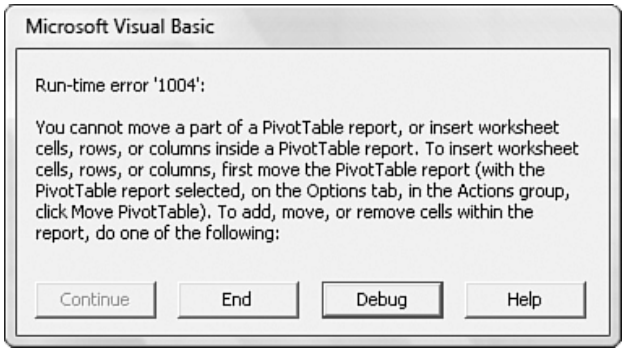

Microsoft Visual Basic
Run-time error '1004':
You cannot move a part of a PivotTable report, or insert worksheet cells, rows, or columns inside a PivotTable report. To insert worksheet cells, rows, or columns, first move the PivotTable report (with the PivotTable report selected, on the Options tab, in the Actions group, click Move PivotTable). To add, move, or remove cells within the report, do one of the following:
Continue
End
Debug
Help

Figure 11.5
You cannot delete just part of a pivot table.

Determining Size of a Finished Pivot Table

Knowing the size of a pivot table in advance is difficult. If you run a report of transactional data on one day, you may or may not have sales from the West region, for example. This could cause your table to be either six or seven columns wide. Therefore, you should use the special property `TableRange2` to refer to the entire resultant pivot table.

Because of the limitations of pivot tables, you should generally copy the results of a pivot table to a new location on the worksheet and then delete the original pivot table. The code in `CreateSummaryReportUsingPivot()` creates a small pivot table. Note that you can set the `ColumnGrand` and `RowGrand` properties of the table to `False` to prevent the totals from being added to the table.

`PT.TableRange2` includes the entire pivot table. In this case, this includes the extra row at the top with the button Sum of Revenue. To eliminate that row, the code copies `PT.TableRange2` but offsets this selection by one row by using `.Offset(1, 0)`. Depending on the nature of your pivot table, you might need to use an offset of two or more rows to get rid of extraneous information at the top of the pivot table.

The code copies `PT.TableRange2` and uses `PasteSpecial` on a cell five rows below the current pivot table. At that point in the code, your worksheet appears as shown in Figure 11.6. The table in R2 is a live pivot table, and the table in R10 is just the copied results.

Figure 11.6
An intermediate result of the macro. Only the summary in R10:V14 will remain after the macro finishes.

	P	Q	R	S	T	U	V
1	SalesRep						
2	6444		Sum of Revenue	Region			
3	160006		Business Segment	WEST	MIDWEST	NORTH	SOUTH
4	160006		Housekeeping and Organization	146623.3443	174518.0799	534284.1862	283170.1652
5	52661		Landscaping and Area Beautification	1236315.87	283042.7327	290612.0313	493495.7632
6	6444		Maintenance and Repair	1621893.009	1183257.128	1106872.882	1876793.789
7	160464						
8	60003						
9	4544						
10	51661		Business Segment	WEST	MIDWEST	NORTH	SOUTH
11	51661		Housekeeping and Organization	146623.3443	174518.0799	534284.1862	283170.1652
12	51631		Landscaping and Area Beautification	1236315.87	283042.7327	290612.0313	493495.7632
13	160614		Maintenance and Repair	1621893.009	1183257.128	1106872.882	1876793.789
14	160614						

You can then totally eliminate the pivot table by applying the `Clear` method to the entire table. If your code is then going on to do additional formatting, you should remove the pivot cache from memory by setting `PTCache` equal to `Nothing`.

The following code will use a pivot table to produce a summary from the underlying data. At the end of the code, the pivot table will be copied to static values and the pivot table will be cleared.

Listing 11.2 Code to Produce a Static Summary from a Pivot Table

```
Sub CreateSummaryReportUsingPivot()
    ' Use a Pivot Table to create a static summary report
    ' with model going down the rows and regions across
    Dim WSD As Worksheet
    Dim PTCache As PivotCache
```

```
Dim PT As PivotTable
Dim PRange As Range
Dim FinalRow As Long
Set WSD = Worksheets("PivotTable")

' Delete any prior pivot tables
For Each PT In WSD.PivotTables
    PT.TableRange2.Clear
Next PT
WSD.Range("R1:AZ1").EntireColumn.Clear

' Define input area and set up a Pivot Cache
FinalRow = WSD.Cells(Rows.Count, 1).End(xlUp).Row
FinalCol = WSD.Cells(1, Columns.Count). _
    End(xlToLeft).Column
Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:= _
    xlDatabase, SourceData:=PRange.Address)

' Create the Pivot Table from the Pivot Cache
Set PT = PTCache.CreatePivotTable(TableDestination:=WSD. _
    Cells(2, FinalCol + 2), TableName:="PivotTable1")

' Turn off updating while building the table
PT.ManualUpdate = True

' Set up the row fields
PT.AddFields RowFields:="Business Segment", ColumnFields:="Region"

' Set up the data fields
With PT.PivotFields("Revenue")
    .Orientation = xlDataField
    .Function = xlSum
    .Position = 1
End With

With PT
    .ColumnGrand = False
    .RowGrand = False
    .NullString = "0"
End With

' Calc the pivot table
PT.ManualUpdate = False
PT.ManualUpdate = True

' PT.TableRange2 contains the results. Move these to R10
' as just values and not a real pivot table.
PT.TableRange2.Offset(1, 0).Copy
WSD.Cells(5 + PT.TableRange2.Rows.Count, FinalCol + 2). _
    PasteSpecial xlPasteValues

' At this point, the worksheet looks like Figure 11.6


' Delete the original Pivot Table & the Pivot Cache
PT.TableRange2.Clear
Set PTCache = Nothing
```

continues

Listing 11.2 Continued

```
    WSD.Activate
    Range("R1").Select
End Sub
```

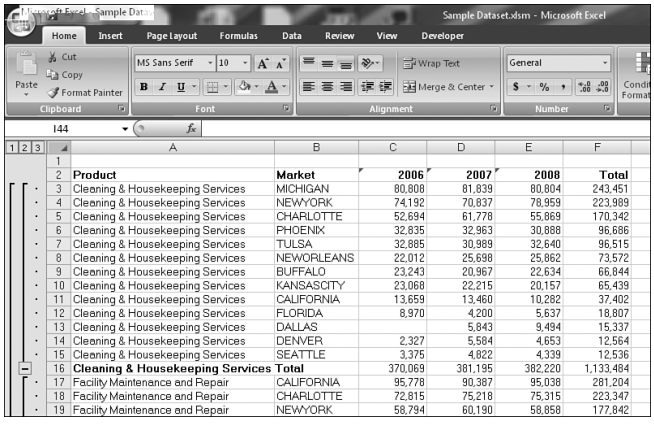
The preceding code creates the pivot table. It then copies the results as values and pastes them as values in R10:V13. Figure 11.6 shows an intermediate result just before the original pivot table is cleared.

So far, this chapter has walked you through building the simplest of pivot table reports. Pivot tables offer far more flexibility. Read on for more complex reporting examples.

Creating a Report Showing Revenue by Product

A typical report might provide a list of markets by product with revenue by year. This report could be given to product line managers to show them which markets are selling well. In this example, you want to show the markets in descending order by revenue with years going across the columns. A sample report is shown in Figure 11.7.

Figure 11.7
A typical request is to take transactional data and produce a summary by product for product line managers. You can use a pivot table to get 90% of this report and then a little formatting to finish it.



	A	B	C	D	E	F
1						
2	**Product**	**Market**	**2006**	**2007**	**2008**	**Total**
3	Cleaning & Housekeeping Services	MICHIGAN	80,808	81,839	80,804	243,451
4	Cleaning & Housekeeping Services	NEWYORK	74,192	70,837	78,959	223,989
5	Cleaning & Housekeeping Services	CHARLOTTE	52,694	61,778	55,869	170,342
6	Cleaning & Housekeeping Services	PHOENIX	32,835	32,963	30,888	96,686
7	Cleaning & Housekeeping Services	TULSA	32,885	30,989	32,640	96,515
8	Cleaning & Housekeeping Services	NEWORLEANS	22,012	25,698	25,862	73,572
9	Cleaning & Housekeeping Services	BUFFALO	23,243	20,967	22,634	66,844
10	Cleaning & Housekeeping Services	KANSASCITY	23,068	22,215	20,157	65,439
11	Cleaning & Housekeeping Services	CALIFORNIA	13,659	13,460	10,282	37,402
12	Cleaning & Housekeeping Services	FLORIDA	8,970	4,200	5,637	18,807
13	Cleaning & Housekeeping Services	DALLAS		5,843	9,494	15,337
14	Cleaning & Housekeeping Services	DENVER	2,327	5,584	4,653	12,564
15	Cleaning & Housekeeping Services	SEATTLE	3,375	4,822	4,339	12,536
16	**Cleaning & Housekeeping Services Total**		370,069	381,195	382,220	1,133,484
17	Facility Maintenance and Repair	CALIFORNIA	95,778	90,387	95,038	281,204
18	Facility Maintenance and Repair	CHARLOTTE	72,815	75,218	75,315	223,347
19	Facility Maintenance and Repair	NEWYORK	58,794	60,190	58,858	177,842

The key to producing this data quickly is to use a pivot table. Although pivot tables are incredible for summarizing data, they are quirky and their presentation is downright ugly. The final result is rarely formatted in a manner that is acceptable to line managers. There is not a good way to insert page breaks between each product in the pivot table.

To create this report, start with a pivot table that has Product and Market as row fields, Invoice Date grouped by year as a column field, and Sum of Revenue as the data field. Figure 11.8 shows the default pivot table created with these settings.

Figure 11.8
Use the power of the pivot table to get the summarized data, but then use your own common sense in formatting the report.

	R	S	T
2	Sum of Rev		InvoiceDate
3	Product	Market	2006
4	⊟Cleaning	BUFFALO	23243.2381
5		CALIFORNIA	13659.459
6		CHARLOTTE	52694.0799
7		DALLAS	8970.1623
8		DENVER	2326.6595
9		FLORIDA	
10		KANSASCITY	23068.2759
11		MICHIGAN	80808.0588
12		NEWORLEANS	22011.5508
13		NEWYORK	74192.2907
14		PHOENIX	32834.6763
15		SEATTLE	3375.0822
16		TULSA	32885.3702
17	Cleaning & Housekeeping S		370068.9037
18	⊟Facility M	BUFFALO	23912.3083
19		CALIFORNIA	95778.1248

Here are just a few of the annoyances that most pivot tables present in their default state:

- The Outline view is horrible. In Figure 11.8, the value Cleaning & Housekeeping Services appears in the product column only once and is followed by 12 blank cells. This is the worst feature of pivot tables, and there is absolutely no way to correct it. Although humans can understand that this entire section is for Cleaning sales, it is radically confusing if your Cleaning section spills to a second or third page. Page 2 starts without any indication that the report is for Cleaning sales. If you intend to repurpose the data, you need the Cleaning sales value to be on every row.
- The report contains blank cells instead of zeros. In Figure 11.8, the Florida market had no cleaning sales in 2006. Excel produces a pivot table where cell T9 is blank instead of zero. This is simply bad form. Excel experts rely on being able to "ride the range," using the End and arrow keys. Blank cells ruin this ability.
- The title is boring. Most people would agree that Sum of Revenue is an annoying title.
- Some captions are extraneous. Invoice Date floating in cell T2 of Figure 11.8 really does not belong in a report.
- The default alphabetical sort order is rarely useful. Product line managers are going to want the top markets at the top of the list. It would be helpful to have the report sorted in descending order by revenue.
- The borders are ugly. Excel draws in a myriad of borders that really make the report look awful.
- The default number format is General. It would be better to set this up as data with commas to serve as thousands separators, or perhaps even data in thousands or millions.
- Pivot tables offer no intelligent page break logic. If you want to be able to produce one report for each Line of Business manager, there is no fast method for indicating that each product should be on a new page.

- Because of the page break problem, you may find it is easier to do away with the pivot table's subtotal rows and have the `Subtotal` method add subtotal rows with page breaks. You need a way to turn off the pivot table subtotal rows offered for Product in Figure 11.8. These rows show up automatically whenever you have two or more row fields. If you had four row fields, you would want to turn off the automatic subtotals for the three outermost row fields.

Even with all these problems in default pivot tables, they are still the way to go. You can overcome each complaint, either by using special settings within the pivot table or by entering a few lines of code after the pivot table is created and then copied to a regular dataset.

Eliminating Blank Cells in the Values Area

People started complaining about the blank cells immediately when pivot tables were first introduced. Anyone using Excel 97 or later can easily replace blank cells with zeros. In the user interface, you can find the setting on the Layout & Format tab of the PivotTable Options dialog box. Choose the For Empty Cells, Show option and type **`0`** in the box.

The equivalent operation in VBA is to set the `NullString` property for the pivot table to `"0"`.

> **NOTE** Although the proper code is to set this value to a text zero, Excel actually puts a real zero in the empty cells.

Ensuring Table Layout Is Utilized

In previous versions of Excel, multiple row fields appeared in multiple columns. Three layouts are now available. The Compact layout squeezes all the row fields into a single column.

To prevent this outcome and ensure that your pivot table is in the classic table layout, use this code:

```
PT.RowAxisLayout xlTabularRow
```

Controlling the Sort Order with AutoSort

The Excel user interface offers an AutoSort option that enables you to show markets in descending order based on revenue. The equivalent code in VBA to sort the product field by descending revenue uses the `AutoSort` method:

```
PT.PivotFields("Market").AutoSort Order:=xlDescending, _
    Field:="Sum of Revenue"
```

Changing Default Number Format

To change the number format in the user interface, choose a revenue field, click PivotTable Tools Options, Active Field, Field Settings, Number Format. Then choose an appropriate number format.

When you have large numbers, displaying the thousands separator helps the person reading the report. To set up this format in VBA code, use the following:

```
PT.PivotFields("Sum of Revenue").NumberFormat = "#,##0"
```

Some companies have customers who typically buy thousands or millions of dollars' worth of goods. You can display numbers in thousands by using a single comma after the number format. Of course, you need to include a `K` abbreviation to indicate that the numbers are in thousands:

```
PT.PivotFields("Sum of Revenue").NumberFormat = "#,##0,K"
```

Of course, local custom dictates the thousands abbreviation. If you are working for a relatively young computer company where everyone uses `K` for the thousands separator, you're in luck because Microsoft makes it easy to use this abbreviation. However, if you work at a 100+ year-old soap company where you use `M` for thousands and `MM` for millions, you have a few more hurdles to jump. You are required to prefix the `M` character with a backslash to have it work:

```
PT.PivotFields("Sum of Revenue").NumberFormat = "#,##0,\M"
```

Alternatively, you can surround the `M` character with double quotation marks. To put double quotation marks inside a quoted string in VBA, you must put two sequential quotation marks. To set up a format in tenths of millions that uses the `#,##0.0,,"MM"` format, you would use this line of code:

```
PT.PivotFields("Sum of Revenue").NumberFormat = "#,##0.0,, ""M"""
```

Here, the format is quotation mark, pound, comma, pound, pound, zero, period, zero, comma, comma, quotation mark, quotation mark, M, quotation mark, quotation mark, quotation mark. The three quotation marks at the end are correct. You use two quotation marks to simulate typing one quotation mark in the custom number format box and a final quotation mark to close the string in VBA.

Suppressing Subtotals for Multiple Row Fields

As soon as you have more than one row field, Excel automatically adds subtotals for all but the innermost row field. However, you may want to suppress subtotals for any number of reasons. Although accomplishing this task manually may be relatively simple, the VBA code to suppress subtotals is surprisingly complex.

You must set the `Subtotals` property equal to an array of 12 `False` values. Read the VBA help for all the gory details, but it goes something like this: The first `False` turns off automatic subtotals, the second `False` turns off the `Sum` subtotal, the third `False` turns off the `Count` subtotal, and so on. It is interesting that you have to turn off all 12 possible subtotals, even though Excel displays only one subtotal. This line of code suppresses the `Product` subtotal:

```
PT.PivotFields("Product").Subtotals = Array(False, False, False, False, _
    False, False, False, False, False, False, False, False)
```

A different technique is to turn on the first subtotal. This method automatically turns off the other 11 subtotals. You can then turn off the first subtotal to make sure that all subtotals are suppressed:

```
PT.PivotFields("Product").Subtotals(1) = True
PT.PivotFields("Product").Subtotals(1) = False
```

Suppressing Grand Total for Rows

Because you are going to be using VBA code to add automatic subtotals, you can get rid of the Grand Total row. If you turn off Grand Total for Rows, you delete the column called Grand Total. Thus, to get rid of the Grand Total row, you must uncheck Grand Total for Columns. This is handled in the code with the following line:

```
PT.ColumnGrand = False
```

Handling Additional Annoyances When Creating Your Final Report

You've reached the end of the adjustments that you can make to the pivot table. To achieve the final report, you have to make the remaining adjustments after converting the pivot table to regular data.

Figure 11.9 shows the pivot table with all the adjustments described in the preceding sections and with `PT.TableRange2` selected.

Figure 11.9
Getting 90% of the way to the final report took less than one second and fewer than 30 lines of code. To solve the last five annoying problems, you have to change this data from a pivot table to regular data.

	R	S	T
2	Sum of Rev		InvoiceDate
3	Product	Market	2006
4	⊟Cleaning	MICHIGAN	81K
5		NEWYORK	74K
6		CHARLOTTE	53K
7		PHOENIX	33K
8		TULSA	33K
9		NEWORLEANS	22K
10		BUFFALO	23K
11		KANSASCITY	23K
12		CALIFORNIA	14K
13		DALLAS	9K
14		FLORIDA	0K
15		DENVER	2K
16		SEATTLE	3K
17	⊟Facility M	CALIFORNIA	96K
18		CHARLOTTE	73K
19		NEWYORK	59K

Creating a New Workbook to Hold the Report

Say you want to build the report in a new workbook so that it can be easily mailed to the product managers. Doing this is fairly easy. To make the code more portable, assign object

variables to the original workbook, new workbook, and first worksheet in the new workbook. At the top of the procedure, add these statements:

```
Dim WSR As Worksheet
Dim WBO As Workbook
Dim WBN As Workbook
Set WBO = ActiveWorkbook
Set WSD = Worksheets("Pivot Table")
```

After the pivot table has been successfully created, build a blank Report workbook with this code:

```
' Create a New Blank Workbook with one Worksheet
Set WBN = Workbooks.Add(xlWorksheet)
Set WSR = WBN.Worksheets(1)
WSR.Name = "Report"
' Set up Title for Report
With WSR.Range("A1")
    .Value = "Revenue by Market and Year"
    .Font.Size = 14
End With
```

Creating a Summary on a Blank Report Worksheet

Imagine that you have submitted the pivot table in Figure 11.9, and your manager hates the borders, hates the title, and hates the words "Invoice Date" in cell T2. You can solve all three of these problems by excluding the first row(s) of `PT.TableRange2` from the `.Copy` method and then using `PasteSpecial(xlPasteValuesAndNumberFormats)` to copy the data to the report sheet.

> **CAUTION**
>
> In Excel 2000 and earlier, `xlPasteValuesAndNumberFormats` was not available. You had to use Paste Special twice: once as `xlPasteValues` and once as `xlPasteFormats`.

In the current example, the `.TableRange2` property includes only one row to eliminate, row 2, as shown in Figure 11.9. If you had a more complex pivot table with several column fields and/or one or more page fields, you would have to eliminate more than just the first row of the report. It helps to run your macro to this point, look at the result, and figure out how many rows you need to delete. You can effectively not copy these rows to the report by using the `Offset` property. Copy the `TableRange2` property, offset by one row. Purists will note that this code copies one extra blank row from below the pivot table, but this really does not matter because the row is blank. After copying, you can erase the original pivot table and destroy the pivot cache:

```
' Copy the Pivot Table data to row 3 of the Report sheet
' Use Offset to eliminate the title row of the pivot table
PT.TableRange2.Offset(1, 0).Copy
WSR. Range("A3").PasteSpecial Paste:=xlPasteValuesAndNumberFormats
PT.TableRange2.Clear
Set PTCache = Nothing
```

Note that you use the Paste Special option to paste just values and number formats. This gets rid of both borders and the pivot nature of the table. You might be tempted to use the All Except Borders option under Paste, but this keeps the data in a pivot table, and you won't be able to insert new rows in the middle of the data.

Filling the Outline View

The report is almost complete. You are nearly a Data, Subtotals command away from having everything you need. Before you can use the Subtotals command, however, you need to fill in all the blank cells in the Outline view of column A.

Fixing the Outline view requires just a few obscure steps. Here are the steps in the user interface:

1. Select all the cells in column A that make up the report.
2. Select Home, Editing, Find & Select, Go To Special to bring up the Go To Special dialog box. Select Blanks to select only the blank cells.
3. Enter an R1C1-style formula to fill the blank with the cell above it. This formula is `=R[1]C`. In the user interface, you would type an equal sign, press the up-arrow key, and then press Ctrl+Enter.
4. Reselect all the cells in column A that make up the report. This step is necessary because the Paste Special step cannot work with noncontiguous selections.
5. Copy the formulas in column A and convert them to values by choosing Home, Clipboard, Paste, Paste Values.

Fixing the Outline view in VBA requires fewer steps. The equivalent VBA logic is shown here:

1. Find the last row of the report.
2. Enter the formula `=R[-1]C` in the blank cells in A.
3. Change those formulas to values. The code to do this follows:

```
Dim FinalReportRow as Long
    ' Fill in the Outline view in column A
    ' Look for last row in column B since many rows
    ' in column A are blank
FinalReportRow = WSR.Cells(Rows.Count, 2).End(xlUp).Row
With Range("A3").Resize(FinalReportRow - 2, 1)
    With .SpecialCells(xlCellTypeBlanks)
        .FormulaR1C1 = "=R[-1]C"
    End With
    .Value = .Value
End With
```

Handling Final Formatting

The last steps for the report involve some basic formatting tasks and then adding the subtotals. You can bold and right-justify the headings in row 3. Set up rows 1–3 so that the top three rows print on each page:

```
' Do some basic formatting
' Autofit columns, bold the headings, right-align
Selection.Columns.AutoFit
Range("A3").EntireRow.Font.Bold = True
Range("A3").EntireRow.HorizontalAlignment = xlRight
Range("A3:B3").HorizontalAlignment = xlLeft

' Repeat rows 1-3 at the top of each page
WSR.PageSetup.PrintTitleRows = "$1:$3"
```

Adding Subtotals

Automatic subtotals are a powerful feature found on the Data menu. Figure 11.10 shows the Subtotal dialog box. Note the option Page Break Between Groups.

If you were sure that you would always have three years and a total, the code to add subtotals for each Line of Business group would be the following:

```
' Add Subtotals by Product.
' Be sure to add a page break at each change in product
Selection.Subtotal GroupBy:=1, Function:=xlSum, TotalList:=Array(3, 4, 5, 6), _
    PageBreaks:=True
```

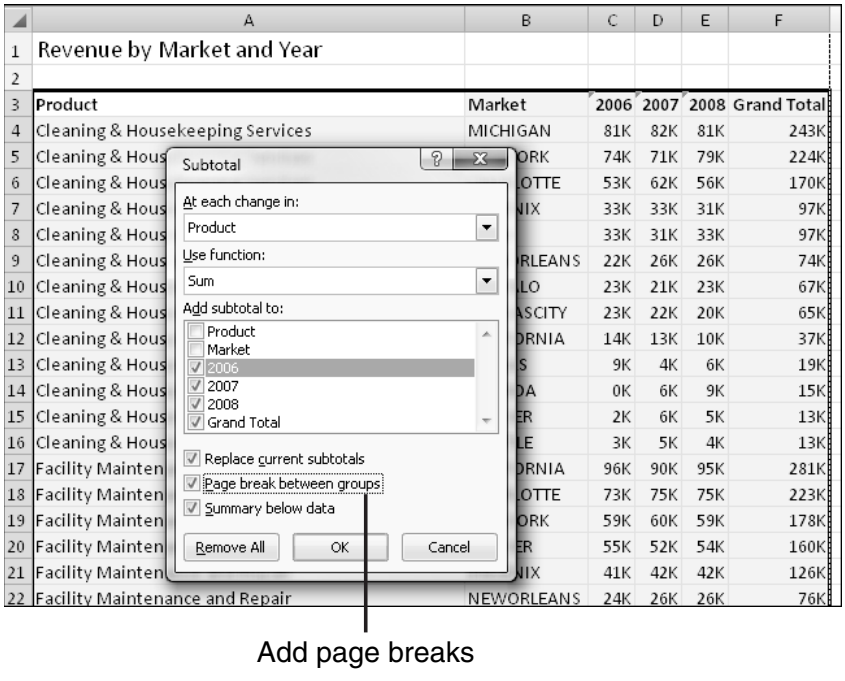


Figure 11.10 Use automatic subtotals because doing so enables you to add a page break after each product. Using this feature ensures that each product manager has a clean report with only her product on it.

However, this code fails if you have more or less than three years. The solution is to use the following convoluted code to dynamically build a list of the columns to total, based on the number of columns in the report:

```
Dim TotColumns()
Dim I as Integer
FinalCol = Cells(3, Columns.Count).End(xlToLeft).Column
ReDim Preserve TotColumns(1 To FinalCol - 2)
For i = 3 To FinalCol
    TotColumns(i - 2) = i
Next i
Selection.Subtotal GroupBy:=1, Function:=xlSum, TotalList:=TotColumns,_
     Replace:=True, PageBreaks:=True, SummaryBelowData:=True
```

Finally, with the new totals added to the report, you need to autofit the numeric columns again with this code:

```
Dim GrandRow as Long
' Make sure the columns are wide enough for totals
GrandRow = Cells(Rows.Count, 1).End(xlUp).Row
Cells(3, 3).Resize(GrandRow - 2, FinalCol - 2).Columns.AutoFit
Cells(GrandRow, 3).Resize(1, FinalCol - 2).NumberFormat = "#,##0,K"
' Add a page break before the Grand Total row, otherwise
' the product manager for the final Line will have two totals
WSR.HPageBreaks.Add Before:=Cells(GrandRow, 1)
```

Putting It All Together

Listing 11.3 produces the product line manager reports in a few seconds.

Listing 11.3 Code That Produces the Product Line Report in Figure 11.11

```
Sub ProductLineReport()
    ' Product and Market as Row
    ' Years as Column
    Dim WSD As Worksheet
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long
    Dim TotColumns()

    Set WSD = Worksheets("PivotTable")
    Dim WSR As Worksheet
    Dim WBO As Workbook
    Dim WBN As Workbook
    Set WBO = ActiveWorkbook

    ' Delete any prior pivot tables
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT
    WSD.Range("R1:AZ1").EntireColumn.Clear

    ' Define input area and set up a Pivot Cache
    FinalRow = WSD.Cells(Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Columns.Count). _
        End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:= _
        xlDatabase, SourceData:=PRange.Address)
```

```
' Create the Pivot Table from the Pivot Cache
Set PT = PTCache.CreatePivotTable(TableDestination:=WSD. _
    Cells(2, FinalCol + 2), TableName:="PivotTable1")

' Turn off updating while building the table
PT.ManualUpdate = True

' Set up the row fields
PT.AddFields RowFields:=Array("Product", "Market"), _
    ColumnFields:="InvoiceDate"
' Ensure table layout, with each row field in a new column
PT.RowAxisLayout xlTabularRow


' Set up the data fields
With PT.PivotFields("Revenue")
    .Orientation = xlDataField
    .Function = xlSum
    .Position = 1
    .NumberFormat = "#,##0,K"
End With

' Calc the pivot table
PT.ManualUpdate = False
PT.ManualUpdate = True

' Group by Year
WSD.Activate
Cells(3, FinalCol + 4).Group Start:=True, End:=True, _
    Periods:=Array(False, False, False, False, False, False, True)

' Replace blanks with zero
PT.NullString = "0"

' Remove subtotals by product
PT.PivotFields("Product").Subtotals(1) = True
PT.PivotFields("Product").Subtotals(1) = False
PT.ColumnGrand = False

' Sort descending by revenue
PT.PivotFields("Market").AutoSort Order:=xlDescending, _
    Field:="Sum of Revenue"

' Calc the pivot table
PT.ManualUpdate = False
PT.ManualUpdate = True

'  PT.TableRange2.Select

' Create a New Blank Workbook with one Worksheet
Set WBN = Workbooks.Add(xlWBATWorksheet)
Set WSR = WBN.Worksheets(1)
WSR.Name = "Report"
' Set up Title for Report
With WSR.[A1]
    .Value = "Revenue by Market and Year"
```

continues

Listing 11.3 Continued

```
        .Font.Size = 14
    End With

    ' Copy the Pivot Table data to row 3 of the Report sheet
    ' Use Offset to eliminate the title row of the pivot table
    PT.TableRange2.Offset(1, 0).Copy
    WSR.[A3].PasteSpecial Paste:=xlPasteValuesAndNumberFormats
    PT.TableRange2.Clear
    Set PTCache = Nothing

    ' Fill in the Outline view in column A
    ' Look for last row in column B since many rows
    ' in column A are blank
    FinalReportRow = WSR.Cells(Rows.Count, 2).End(xlUp).Row
    With Range("A3").Resize(FinalReportRow - 2, 1)
        With .SpecialCells(xlCellTypeBlanks)
            .FormulaR1C1 = "=R[-1]C"
        End With
        .Value = .Value
    End With

    ' Do some basic formatting
    ' Autofit columns, bold the headings, right-align
    Selection.Columns.AutoFit
    Range("A3").EntireRow.Font.Bold = True
    Range("A3").EntireRow.HorizontalAlignment = xlRight
    Range("A3:B3").HorizontalAlignment = xlLeft

    ' Repeat rows 1-3 at the top of each page
    WSR.PageSetup.PrintTitleRows = "$1:$3"

    ' Add subtotals
    FinalCol = Cells(3, Columns.Count).End(xlToLeft).Column
    ReDim Preserve TotColumns(1 To FinalCol - 2)
    For i = 3 To FinalCol
        TotColumns(i - 2) = i
    Next i
    Selection.Subtotal GroupBy:=1, Function:=xlSum, _
        TotalList:=TotColumns, Replace:=True, _
        PageBreaks:=True, SummaryBelowData:=True

    ' Make sure the columns are wide enough for totals
    GrandRow = WSR.Cells(Rows.Count, 1).End(xlUp).Row
    Cells(3, 3).Resize(GrandRow - 2, FinalCol - 2).Columns.AutoFit
    Cells(GrandRow, 3).Resize(1, FinalCol - 2).NumberFormat = "#,##0,K"
    ' Add a page break before the Grand Total row, otherwise
    ' the product manager for the final Line will have two totals
    WSR.HPageBreaks.Add Before:=Cells(GrandRow, 1)

End Sub
```

Figure 11.11 shows the report produced by this code.

Figure 11.11
Converting 50,000 rows of transactional data to this useful report takes less than two seconds if you use the code that produced this example. Without pivot tables, the code would be far more complex.

	A	B	C	D	E	F
1	Revenue by Market and Year					
2						
3	**Product**	**Market**	**2006**	**2007**	**2008**	**Grand Total**
4	Cleaning & Housekeeping Services	MICHIGAN	81K	82K	81K	243K
5	Cleaning & Housekeeping Services	NEWYORK	74K	71K	79K	224K
6	Cleaning & Housekeeping Services	CHARLOTTE	53K	62K	56K	170K
7	Cleaning & Housekeeping Services	PHOENIX	33K	33K	31K	97K
8	Cleaning & Housekeeping Services	TULSA	33K	31K	33K	97K
9	Cleaning & Housekeeping Services	NEWORLEANS	22K	26K	26K	74K
10	Cleaning & Housekeeping Services	BUFFALO	23K	21K	23K	67K
11	Cleaning & Housekeeping Services	KANSASCITY	23K	22K	20K	65K
12	Cleaning & Housekeeping Services	CALIFORNIA	14K	13K	10K	37K
13	Cleaning & Housekeeping Services	DALLAS	9K	4K	6K	19K
14	Cleaning & Housekeeping Services	FLORIDA	0K	6K	9K	15K
15	Cleaning & Housekeeping Services	DENVER	2K	6K	5K	13K
16	Cleaning & Housekeeping Services	SEATTLE	3K	5K	4K	13K
17	**Cleaning & Housekeeping Services Total**		370K	381K	382K	1,133K
18	Facility Maintenance and Repair	CALIFORNIA	96K	90K	95K	281K
19	Facility Maintenance and Repair	CHARLOTTE	73K	75K	75K	223K

Addressing Issues with Two or More Data Fields

So far, you have built some powerful summary reports, but you've touched only a portion of the powerful features available in pivot tables. The preceding example produced a report but had only one data field. It is possible to have multiple fields in the Σ Values section of a pivot report. The data in this example includes not just revenue, but also a count of customers.

When you have two or more data fields, you have a choice of placing the data fields in one of four locations. By default, Excel builds the pivot report with the data field as the innermost column field. It is often preferable to have the data field as the outermost row field.

When a pivot table is going to have more than one data field, you have a virtual field named Σ Values in the drop zones of the PivotTable Field List. In VBA, this equivalent virtual field is named Data.

Where you place the data field in the `.AddFields` method determines which view of the data you get. The default setup, with the data fields arranged as the innermost column field, as shown in Figure 11.12, would have this `AddFields` line:

```
PT.AddFields ColumnFields:=Array("Region", "Data")
```

The view shown in Figure 11.13 would use this code:

```
PT.AddFields RowFields:=Array("Data", "Product"")
```

One view that would make sense would have Data as the only column field:

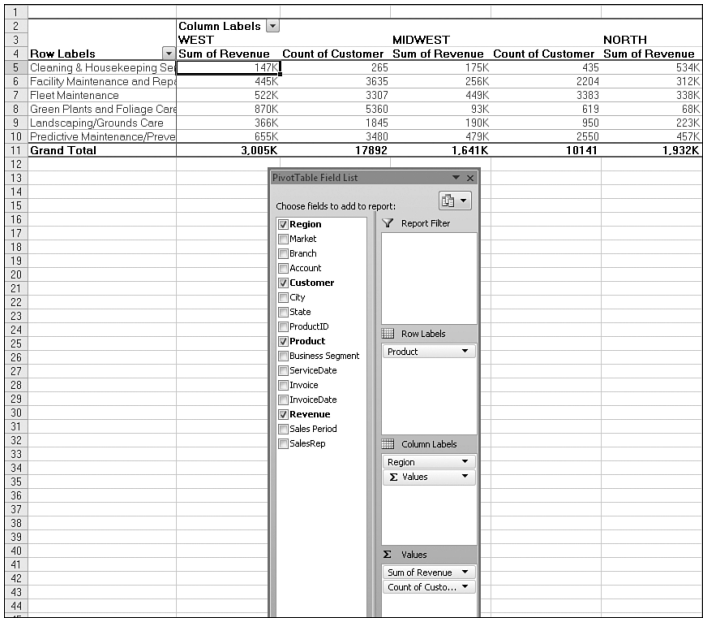
```
PT.AddFields RowFields:="Product", ColumnFields:="Data"
```

After adding a column field called Data, you would then go on to define two data fields:

```
' Set up the data fields
With PT.PivotFields("Revenue")
    .Orientation = xlDataField
    .Function = xlSum
    .Position = 1
    .NumberFormat = "#,##0,K"
End With
```

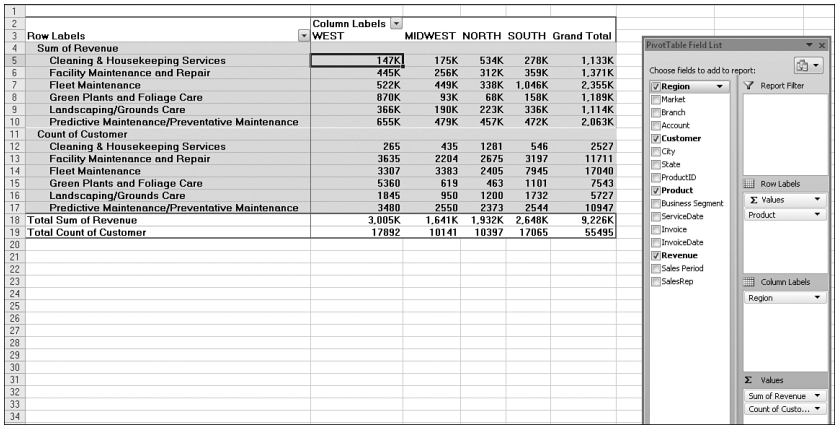
```
With PT.PivotFields("Units")
    .Orientation = xlDataField
    .Function = xlSum
    .Position = 2
    .NumberFormat = "#,##0"
End With
```

Figure 11.12
The default pivot table report has multiple data fields as the innermost column field.



Row Labels	WEST Sum of Revenue	WEST Count of Customer	MIDWEST Sum of Revenue	MIDWEST Count of Customer	NORTH Sum of Revenue
Cleaning & Housekeeping Se	147K	265	175K	435	534K
Facility Maintenance and Repa	445K	3635	256K	2204	312K
Fleet Maintenance	522K	3307	449K	3383	338K
Green Plants and Foliage Care	870K	5360	93K	619	68K
Landscaping/Grounds Care	366K	1845	190K	950	223K
Predictive Maintenance/Preve	655K	3480	479K	2550	457K
Grand Total	**3,005K**	**17892**	**1,641K**	**10141**	**1,932K**

Figure 11.13
By moving the data field to the first row field, you can obtain this view of the multiple data fields.



Row Labels	WEST	MIDWEST	NORTH	SOUTH	Grand Total
Sum of Revenue					
Cleaning & Housekeeping Services	147K	175K	534K	278K	1,133K
Facility Maintenance and Repair	445K	256K	312K	359K	1,371K
Fleet Maintenance	522K	449K	338K	1,046K	2,355K
Green Plants and Foliage Care	870K	93K	68K	158K	1,189K
Landscaping/Grounds Care	366K	190K	223K	336K	1,114K
Predictive Maintenance/Preventative Maintenance	655K	479K	457K	472K	2,063K
Count of Customer					
Cleaning & Housekeeping Services	265	435	1281	546	2527
Facility Maintenance and Repair	3635	2204	2675	3197	11711
Fleet Maintenance	3307	3383	2405	7945	17040
Green Plants and Foliage Care	5360	619	463	1101	7543
Landscaping/Grounds Care	1845	950	1200	1732	5727
Predictive Maintenance/Preventative Maintenance	3480	2550	2373	2544	10947
Total Sum of Revenue	3,005K	1,641K	1,932K	2,648K	9,226K
Total Count of Customer	17892	10141	10397	17065	55495

Calculated Data Fields

Pivot tables offer two types of formulas. The most useful type defines a formula for a calculated field. This adds a new field to the pivot table. Calculations for calculated fields are

always done at the summary level. If you define a calculated field for average price as Revenue divided by Units Sold, Excel first adds the total revenue and total quantity, and then it does the division of these totals to get the result. In many cases, this is exactly what you need. If your calculation does not follow the associative law of mathematics, it might not work as you expect.

To set up a calculated field, use the `Add` method with the `CalculatedFields` object. You have to specify a field name and a formula. Note that if you create a field called Average Price, the default pivot table produces a field called Sum of Average Price. This title is misleading and downright silly. What you have is actually the average of the sums of prices. The solution is to use the `Name` property when defining the data field to replace Sum of Average Price with something such as Avg Price. Note that this name must be different from the name for the calculated field.

Listing 11.4 produces the report shown in Figure 11.14.

Listing 11.4 Code That Calculates an Average Price Field as a Second Data Field

```
Sub TwoDataFields()
    ' Listing 11.4
    Dim WSD As Worksheet
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long

    Set WSD = Worksheets("PivotTable")
    Dim WSR As Worksheet
    Dim WBO As Workbook
    Dim WBN As Workbook
    Set WBO = ActiveWorkbook

    ' Delete any prior pivot tables
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT
    WSD.Range("R1:AZ1").EntireColumn.Clear

    ' Define input area and set up a Pivot Cache
    FinalRow = WSD.Cells(Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Columns.Count). _
        End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:= _
        xlDatabase, SourceData:=PRange.Address)

    ' Create the Pivot Table from the Pivot Cache
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD. _
        Cells(2, FinalCol + 2), TableName:="PivotTable1")

    ' Turn off updating while building the table
    PT.ManualUpdate = True
```

continues

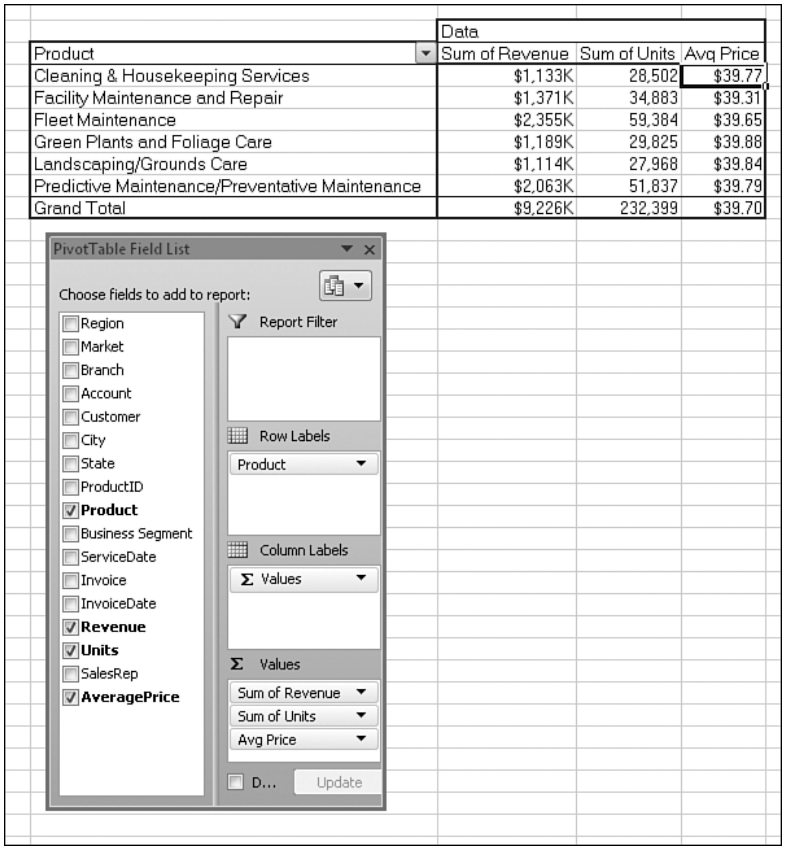
Listing 11.4 Continued

```
    ' Set up the row fields
    PT.AddFields RowFields:="Product", ColumnFields:="Data"

    ' Define Calculated Fields
    PT.CalculatedFields.Add Name:="AveragePrice", Formula:="=Revenue/Units"

    ' Set up the data fields
    With PT.PivotFields("Revenue")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
        .NumberFormat = "$#,##0,K"
    End With

    With PT.PivotFields("Units")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 2
        .NumberFormat = "#,##0"
    End With

    With PT.PivotFields("AveragePrice")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 3
        .NumberFormat = "$#,##0.00"
        .Name = "Avg Price"
    End With

    ' Ensure that we get zeros instead of blanks in the data area
    PT.NullString = "0"

    ' Calc the pivot table
    PT.ManualUpdate = False
    PT.ManualUpdate = True

    WSD.Activate
    Range("R1").Select

End Sub
```


Calculated Items

Say that in your company one manager is responsible for Landscaping/Grounds Care and Green Plants and Foliage Care. The idea behind a calculated item is that you can define a new item along the Product field to calculate the total of these two items. Listing 11.5 produces the report shown in Figure 11.15.

Figure 11.14
The virtual Data dimension contains two fields from your dataset plus a calculation. It is shown along the column area of the report.



	Data		
Product	Sum of Revenue	Sum of Units	Avg Price
Cleaning & Housekeeping Services	$1,133K	28,502	$39.77
Facility Maintenance and Repair	$1,371K	34,883	$39.31
Fleet Maintenance	$2,355K	59,384	$39.65
Green Plants and Foliage Care	$1,189K	29,825	$39.88
Landscaping/Grounds Care	$1,114K	27,968	$39.84
Predictive Maintenance/Preventative Maintenance	$2,063K	51,837	$39.79
Grand Total	$9,226K	232,399	$39.70

Listing 11.5 Code That Adds a New Item Along the Product Dimension

```
Sub CalcItemsProblem()
    ' Listing 11.5
    Dim WSD As Worksheet
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long

    Set WSD = Worksheets("PivotTable")
    Dim WSR As Worksheet

    ' Delete any prior pivot tables
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT
    WSD.Range("R1:AZ1").EntireColumn.Clear
```

continues

Listing 11.5 Continued

```
    ' Define input area and set up a Pivot Cache
    FinalRow = WSD.Cells(Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Columns.Count). _
        End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:= _
        xlDatabase, SourceData:=PRange.Address)

    ' Create the Pivot Table from the Pivot Cache
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD. _
        Cells(2, FinalCol + 2), TableName:="PivotTable1")

    ' Turn off updating while building the table
    PT.ManualUpdate = True

    ' Set up the row fields
    PT.AddFields RowFields:="Product"

    ' Define calculated item along the product dimension
    PT.PivotFields("Product").CalculatedItems _
        .Add "Plants Group", _
        "='Landscaping/Grounds Care'+'Green Plants and Foliage Care'"

    ' Resequence so that the report Landscaping First
    PT.PivotFields("Product"). _
        PivotItems("Landscaping/Grounds Care").Position = 1
    PT.PivotFields("Product"). _
        PivotItems("Green Plants and Foliage Care").Position = 2
    PT.PivotFields("Product"). _
        PivotItems("Plants Group").Position = 3

    ' Set up the data fields
    With PT.PivotFields("Revenue")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
        .NumberFormat = "#,##0"
    End With

    ' Ensure that we get zeros instead of blanks in the data area
    PT.NullString = "0"

    ' Calc the pivot table
    PT.ManualUpdate = False
    PT.ManualUpdate = True
    WSD.Activate
    Range("R1").Select

End Sub
```

Figure 11.15
Unless you love restating numbers to the Securities and Exchange Commission, avoid using calculated items.

Sum of Revenue	
Product	Total
Landscaping/Grounds Care	1,114,160
Green Plants and Foliage Care	1,189,306
Plants Group	2,303,466
Cleaning & Housekeeping Services	1,133,484
Facility Maintenance and Repair	1,371,359
Fleet Maintenance	2,354,623
Predictive Maintenance/Preventative Maintenance	2,062,835
Grand Total	11,529,233

Look closely at the results shown in Figure 11.15. The calculation for Plants Group is correct. The approximate $2.3 million for the Plants Group is the sum of $1.1 million of landscaping and $1.2 million of Green Plants. However, the grand total should be about $9.2 million. Instead, Excel gives you a grand total of $11.6 million. The total revenue for the company just increased by $2.3 million. Excel gives the wrong grand total when a field contains both regular and calculated items. The only plausible method for dealing with this situation is to attempt to hide the products that make up the Plants Group:

```
With PT.PivotFields("Product")
    .PivotItems("Landscaping/Grounds Care").Visible = False
    .PivotItems("Green Plants and Foliage Care").Visible = False
End With
```

The results are shown in Figure 11.16.

Figure 11.16
After the components that make up the calculated Plants Group item are hidden, the total revenue for the company is again correct. However, it would be easier to add a new field to the original data with a Responsibility field.

Sum of Revenue	
Product	Total
Plants Group	2,303,466
Cleaning & Housekeeping Services	1,133,484
Facility Maintenance and Repair	1,371,359
Fleet Maintenance	2,354,623
Predictive Maintenance/Preventative Maintenance	2,062,835
Grand Total	9,225,767

Summarizing Date Fields with Grouping

With transactional data, you often find your date-based summaries having one row per day. Although daily data might be useful to a plant manager, many people in the company want to see totals by month or quarter and year.

The great news is that Excel handles the summarization of dates in a pivot table with ease. For anyone who has ever had to use the arcane formula `=A2+1-Day(A2)` to change daily dates into monthly dates, you will appreciate the ease with which you can group transactional data into months or quarters.

Creating a group with VBA is a bit quirky. The `.Group` method can be applied to only a single cell in the pivot table, and that cell must contain a date or the Date field label. This is the first example in this chapter where you must allow VBA to calculate an intermediate pivot table result.

You must define a pivot table with Invoice Date in the row field. Turn off `ManualCalculation` to allow the Date field to be drawn. You can then use the `LabelRange` property to locate the date label and group from there. Figure 11.17 shows the result of Listing 11.6.

Figure 11.17
The In Balance Date field is now composed of three fields in the pivot table, representing year, quarter, and month.

Sum of Revenue			Region		
Years	Quarters	InvoiceDate	WEST	MIDWEST	N
⊟2006	⊟Qtr1	Jan	67,576	30,920	
		Feb	104,759	59,787	
		Mar	65,539	34,845	
	⊟Qtr2	Apr	87,919	51,608	
		May	89,887	51,192	
		Jun	82,794	44,692	
	⊟Qtr3	Jul	92,481	50,590	
		Aug	93,391	45,958	
		Sep	78,439	42,088	

Listing 11.6 Code That Uses the Group Feature to Roll Daily Dates Up to Monthly Dates

```
Sub ReportByMonth()
    ' Listing 11.6
    Dim WSD As Worksheet
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long

    Set WSD = Worksheets("PivotTable")
    Dim WSR As Worksheet

    ' Delete any prior pivot tables
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT
    WSD.Range("R1:AZ1").EntireColumn.Clear

    ' Define input area and set up a Pivot Cache
    FinalRow = WSD.Cells(Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Columns.Count). _
        End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:= _
        xlDatabase, SourceData:=PRange.Address)

    ' Create the Pivot Table from the Pivot Cache
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD. _
        Cells(2, FinalCol + 2), TableName:="PivotTable1")

    ' Turn off updating while building the table
    PT.ManualUpdate = True
```

```
    ' Set up the row fields
    PT.AddFields RowFields:="InvoiceDate", ColumnFields:="Region"

    ' Set up the data fields
    With PT.PivotFields("Revenue")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
        .NumberFormat = "#,##0"
    End With

    ' Ensure that we get zeros instead of blanks in the data area
    PT.NullString = "0"

    ' Calc the pivot table to allow the date label to be drawn
    PT.ManualUpdate = False
    PT.ManualUpdate = True
    WSD.Activate

    ' Group ShipDate by Month, Quarter, Year
    PT.PivotFields("InvoiceDate").LabelRange.Group Start:=True, _
        End:=True, Periods:= _
        Array(False, False, False, False, True, True, True)

    ' Calc the pivot table
    PT.ManualUpdate = False
    PT.ManualUpdate = True
    WSD.Activate
    Range("R1").Select

End Sub
```


Group by Week

You probably noticed that Excel allows you to group by day, month, quarter, and year. There is no standard grouping for week. You can, however, define a group that bunches groups of seven days.

By default, Excel starts the week based on the first date found in the data. This means that the default week would run from Thursday, January 5, 2006, through Wednesday, December 31, 2008. You can override this by changing the `Start` parameter from `True` to an actual date. Use the `WeekDay` function to determine how many days to adjust the start date.

There is one limitation to grouping by week. When you group by week, you cannot also group by any other measure. Grouping by week and quarter is not valid.

Listing 11.7 creates the report shown in Figure 11.18.

Figure 11.18
Use the Number of Days setting to group by week.

Sum of Revenue	Region	
InvoiceDate	WEST	MIDWEST
1/2/2006 - 1/8/2006	9,828	5,513
1/9/2006 - 1/15/2006	20,405	9,189
1/16/2006 - 1/22/2006	14,013	8,017
1/23/2006 - 1/29/2006	15,758	5,563
1/30/2006 - 2/5/2006	18,759	9,025
2/6/2006 - 2/12/2006	29,668	16,976
2/13/2006 - 2/19/2006	26,972	14,577
2/20/2006 - 2/26/2006	26,097	15,591

Listing 11.7 Code That Uses the Group Feature to Roll Daily Dates Up to Weekly Dates

```
Sub ReportByWeek()
    ' Listing 11.7
    Dim WSD As Worksheet
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long

    Set WSD = Worksheets("PivotTable")
    Dim WSR As Worksheet

    ' Delete any prior pivot tables
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT
    WSD.Range("R1:AZ1").EntireColumn.Clear

    ' Define input area and set up a Pivot Cache
    FinalRow = WSD.Cells(Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Columns.Count). _
        End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:= _
        xlDatabase, SourceData:=PRange.Address)

    ' Create the Pivot Table from the Pivot Cache
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD. _
        Cells(2, FinalCol + 2), TableName:="PivotTable1")

    ' Turn off updating while building the table
    PT.ManualUpdate = True

    ' Set up the row fields
    PT.AddFields RowFields:="InvoiceDate", ColumnFields:="Region"

    ' Set up the data fields
    With PT.PivotFields("Revenue")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
        .NumberFormat = "#,##0"
    End With
```

```
    ' Ensure that we get zeros instead of blanks in the data area
    PT.NullString = "0"

    ' Calc the pivot table to allow the date label to be drawn
    PT.ManualUpdate = False
    PT.ManualUpdate = True
    WSD.Activate

    ' Group Date by Week.
    'Figure out the first Monday before the minimum date
    FirstDate = Application.Min(PT.PivotFields("InvoiceDate").DataRange)
    WhichDay = Application.WorksheetFunction.Weekday(FirstDate, 3)
    StartDate = FirstDate - WhichDay
    PT.PivotFields("InvoiceDate").LabelRange.Group _
        Start:=StartDate, End:=True, By:=7, _
        Periods:=Array(False, False, False, True, False, False, False)

    ' Calc the pivot table
    PT.ManualUpdate = False
    PT.ManualUpdate = True
    WSD.Activate
    Range("R1").Select

End Sub
```


Using Advanced Pivot Table Techniques

You may be a pivot table pro and never have run into some of the really advanced techniques available with pivot tables. The following sections discuss such techniques.

Using `AutoShow` to Produce Executive Overviews

If you are designing an executive dashboard utility, you might want to spotlight the top five markets.

As with the AutoSort option, you could be a pivot table pro and never have stumbled across the AutoShow feature in Excel. This setting lets you select either the top or bottom *n* records based on any data field in the report.

The code to use AutoShow in VBA uses the `.AutoShow` method:

```
' Show only the top 5 Markets
PT.PivotFields("Market").AutoShow Top:=xlAutomatic, Range:=xlTop, _
    Count:=5, Field:= "Sum of Revenue"
```

When you create a report using the `.AutoShow` method, it is often helpful to copy the data and then go back to the original pivot report to get the totals for all markets. In the following code, this is achieved by removing the Market field from the pivot table and copying the grand total to the report. Listing 11.8 produces the report shown in Figure 11.19.

Figure 11.19
The Top 5 Markets report contains two pivot tables.

	A	B	C	D	E	F	G	H
1	Top 5 Markets							
2								
3	Market	Cleaning & Housekeeping Services	Facility Maintenance and Repair	Fleet Maintenance	Green Plants and Foliage Care	Landscaping/ Grounds Care	Predictive Maintenance /Preventative Maintenance	Grand Total
4	CALIFORNIA	37,402	281,204	337,225	830,422	248,343	520,156	2,254,752
5	FLORIDA	15,337	0	556,004	85,134	153,011	225,761	1,035,246
6	CHARLOTTE	170,342	223,347	245,120	46,483	80,240	124,983	890,514
7	NEWYORK	223,989	177,842	184,747	24,049	75,804	187,150	873,581
8	DENVER	12,564	160,325	170,188	42,408	73,622	186,477	645,584
9	Top 5 Total	459,633	842,718	1,493,284	1,028,496	631,020	1,244,527	5,699,677
10								
11	Total Company	1,133,484	1,371,359	2,354,623	1,189,306	1,114,160	2,062,835	9,225,767

Listing 11.8 Code Used to Create the Top 5 Markets Report

```
Sub Top5Markets()
    ' Listing 11.8
    ' Produce a report of the top 5 markets
    Dim WSD As Worksheet
    Dim WSR As Worksheet
    Dim WBN As Workbook
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long
    Set WSD = Worksheets("PivotTable")

    ' Delete any prior pivot tables
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT
    WSD.Range("R1:AZ1").EntireColumn.Clear

    ' Define input area and set up a Pivot Cache
    FinalRow = WSD.Cells(Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Columns.Count). _
        End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:= _
        xlDatabase, SourceData:=PRange.Address)

    ' Create the Pivot Table from the Pivot Cache
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD. _
        Cells(2, FinalCol + 2), TableName:="PivotTable1")

    ' Turn off updating while building the table
    PT.ManualUpdate = True

    ' Set up the row fields
    PT.AddFields RowFields:="Market", ColumnFields:="Product"

    ' Set up the data fields
    With PT.PivotFields("Revenue")
        .Orientation = xlDataField
        .Function = xlSum
```

```
        .Position = 1
        .NumberFormat = "#,##0"
        .Name = "Total Revenue"
    End With

    ' Ensure that we get zeros instead of blanks in the data area
    PT.NullString = "0"

    ' Sort markets descending by sum of revenue
    PT.PivotFields("Market").AutoSort Order:=xlDescending, _
        Field:="Total Revenue"

    ' Show only the top 5 markets
    PT.PivotFields("Market").AutoShow Type:=xlAutomatic, Range:=xlTop, _
        Count:=5, Field:="Total Revenue"

    ' Calc the pivot table to allow the date label to be drawn
    PT.ManualUpdate = False
    PT.ManualUpdate = True

    ' Create a new blank workbook with one worksheet
    Set WBN = Workbooks.Add(xlWBATWorksheet)
    Set WSR = WBN.Worksheets(1)
    WSR.Name = "Report"
    ' Set up title for report
    With WSR.[A1]
        .Value = "Top 5 Markets"
        .Font.Size = 14
    End With

    ' Copy the pivot table data to row 3 of the report sheet
    ' Use offset to eliminate the title row of the pivot table
    PT.TableRange2.Offset(1, 0).Copy
    WSR.[A3].PasteSpecial Paste:=xlPasteValuesAndNumberFormats
    LastRow = WSR.Cells(65536, 1).End(xlUp).Row
    WSR.Cells(LastRow, 1).Value = "Top 5 Total"

    ' Go back to the pivot table to get totals without the AutoShow
    PT.PivotFields("Market").Orientation = xlHidden
    PT.ManualUpdate = False
    PT.ManualUpdate = True
    PT.TableRange2.Offset(2, 0).Copy
    WSR.Cells(LastRow + 2, 1).PasteSpecial Paste:=xlPasteValuesAndNumberFormats
    WSR.Cells(LastRow + 2, 1).Value = "Total Company"

    ' Clear the pivot table
    PT.TableRange2.Clear
    Set PTCache = Nothing

    ' Do some basic formatting
    ' Autofit columns, bold the headings, right-align
    WSR.Range(WSR.Range("A3"), WSR.Cells(LastRow + 2, 10)).Columns.AutoFit
    Range("A3").EntireRow.Font.Bold = True
    Range("A3").EntireRow.HorizontalAlignment = xlRight
    Range("A3").HorizontalAlignment = xlLeft

    Range("A2").Select
    MsgBox "CEO Report has been Created"
End Sub
```

The Top 5 Markets report actually contains two snapshots of a pivot table. After using the AutoShow feature to grab the top five markets with their totals, the macro went back to the pivot table, removed the AutoShow option, and grabbed the total of all markets to produce the Total Company row.

Using `ShowDetail` to Filter a Recordset

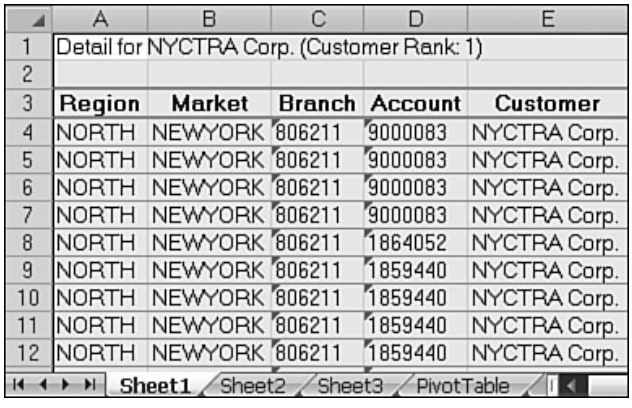
Take any pivot table in the Excel user interface. Double-click any number in the table. Excel inserts a new sheet in the workbook and copies all the source records that represent that number. In the Excel user interface, this is a great way to perform a drill-down query into a dataset.

The equivalent VBA property is `ShowDetail`. By setting this property to `True` for any cell in the pivot table, you generate a new worksheet with all the records that make up that cell:

```
PT.TableRange2.Offset(2, 1).Resize(1, 1).ShowDetail = True
```

Listing 11.9 produces a pivot table with the total revenue for the top three stores and `ShowDetail` for each of those stores. This is an alternative method to using the Advanced Filter report. The results of this macro are three new sheets. Figure 11.20 shows the first sheet created.

Figure 11.20
Pivot table applications are incredibly diverse. This macro created a pivot table of the top three stores and then used the `ShowDetail` property to retrieve the records for each of those stores.



	A	B	C	D	E
1	Detail for NYCTRA Corp. (Customer Rank: 1)				
2					
3	Region	Market	Branch	Account	Customer
4	NORTH	NEWYORK	806211	9000083	NYCTRA Corp.
5	NORTH	NEWYORK	806211	9000083	NYCTRA Corp.
6	NORTH	NEWYORK	806211	9000083	NYCTRA Corp.
7	NORTH	NEWYORK	806211	9000083	NYCTRA Corp.
8	NORTH	NEWYORK	806211	1864052	NYCTRA Corp.
9	NORTH	NEWYORK	806211	1859440	NYCTRA Corp.
10	NORTH	NEWYORK	806211	1859440	NYCTRA Corp.
11	NORTH	NEWYORK	806211	1859440	NYCTRA Corp.
12	NORTH	NEWYORK	806211	1859440	NYCTRA Corp.

Sheet1 Sheet2 Sheet3 PivotTable

Listing 11.9 Code Used to Create a Report for Each of the Top 3 Customers

```
Sub RetrieveTop3CustomerDetail()
    ' Listing 11.9
    ' Retrieve Details from Top 3 Stores
    Dim WSD As Worksheet
    Dim WSR As Worksheet
    Dim WBN As Workbook
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long
    Set WSD = Worksheets("PivotTable")
```

```
' Delete any prior pivot tables
For Each PT In WSD.PivotTables
    PT.TableRange2.Clear
Next PT
WSD.Range("R1:AZ1").EntireColumn.Clear

' Define input area and set up a Pivot Cache
FinalRow = WSD.Cells(Rows.Count, 1).End(xlUp).Row
FinalCol = WSD.Cells(1, Columns.Count). _
    End(xlToLeft).Column
Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:= _
    xlDatabase, SourceData:=PRange.Address)

' Create the Pivot Table from the Pivot Cache
Set PT = PTCache.CreatePivotTable(TableDestination:=WSD. _
    Cells(2, FinalCol + 2), TableName:="PivotTable1")

' Turn off updating while building the table
PT.ManualUpdate = True

' Set up the row fields
PT.AddFields RowFields:="Customer", ColumnFields:="Data"

' Set up the data fields
With PT.PivotFields("Revenue")
    .Orientation = xlDataField
    .Function = xlSum
    .Position = 1
    .NumberFormat = "#,##0"
    .Name = "Total Revenue"
End With

' Sort Stores descending by sum of revenue
PT.PivotFields("Customer").AutoSort Order:=xlDescending, _
    Field:="Total Revenue"

' Show only the top 3 stores
PT.PivotFields("Customer").AutoShow Type:=xlAutomatic, Range:=xlTop, _
    Count:=3, Field:="Total Revenue"

' Ensure that we get zeros instead of blanks in the data area
PT.NullString = "0"

' Calc the pivot table to allow the date label to be drawn
PT.ManualUpdate = False
PT.ManualUpdate = True

' Produce summary reports for each customer
For i = 1 To 3
    PT.TableRange2.Offset(i + 1, 1).Resize(1, 1).ShowDetail = True
    ' The active sheet has changed to the new detail report
    ' Add a title
    Range("A1:A2").EntireRow.Insert
    Range("A1").Value = "Detail for " & _
        PT.TableRange2.Offset(i + 1, 0).Resize(1, 1).Value & _
        " (Customer Rank: " & i & ")"
Next i
```

continues

Listing 11.9 Continued

```
    MsgBox "Detail reports for top 3 customers have been created."
End Sub
```

Creating Reports for Each Region or Model

A pivot table can have one or more Report Filter fields. A Report Filter field goes in a separate set of rows above the pivot report. It can serve to filter the report to a certain region, certain model, or certain combination of region and model.

In VBA, Report Filter fields are called *page fields*.

To set up a page field in VBA, add the `PageFields` parameter to the `AddFields` method. The following line of code creates a pivot table with Region in the page field:

```
PT.AddFields RowFields:= "Product", ColumnFields:= "Data", PageFields:= "Region"
```

The preceding line of code sets up the Region page field with the value (All), which returns all regions. To limit the report to just the North region, use the `CurrentPage` property:

```
PT.PivotFields("Region").CurrentPage = "North"
```


One use of a page field is to build a user form in which someone can select a particular region or particular product. You then use this information to set the `CurrentPage` property and display the results of the user form.

Another interesting use is to loop through all `PivotItems` and display them one at a time in the page field. You can quickly produce top 10 reports for each region using this method.

To determine how many regions are available in the data, use `PT.PivotFields("Region").PivotItems.Count`. Either of these loops would work:

```
For i = 1 To PT.PivotFields("Region").PivotItems.Count
    PT.PivotFields("Region").CurrentPage = _
            PT.PivotFields("Region").PivotItems(i).Name
    PT.ManualUpdate = False
    PT.ManualUpdate = True
Next i

For Each PivItem In PT.PivotFields("Region").PivotItems
    PT.PivotFields("Region").CurrentPage = PivItem.Name
    PT.ManualUpdate = False
    PT.ManualUpdate = True
Next PivItem
```

Of course, in both of these loops, the three region reports fly by too quickly to see. In practice, you would want to save each report while it is displayed.

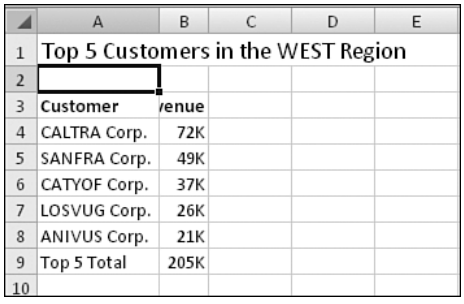
So far in this chapter, you have been using `PT.TableRange2` when copying the data from the pivot table. The `TableRange2` property includes all rows of the pivot table, including the page fields. There is also a `.TableRange1` property, which excludes the page fields. You can use either statement to get the detail rows:

```
PT.TableRange2.Offset(3, 0)
PT.TableRange1.Offset(1, 0)
```

Which you use is your preference, but if you use `TableRange2`, you won't have problems when you try to delete the pivot table with `PT.TableRange2.Clear`. If you were to accidentally attempt to clear `TableRange1` when there are page fields, you would end up with the dreaded "Cannot move or change part of a pivot table" error.

Listing 11.10 produces a new workbook for each region, as shown in Figure 11.21.

Figure 11.21
By looping through all items found in the Region page field, the macro produced one workbook for each regional manager.



	A	B	C	D	E
1	Top 5 Customers in the WEST Region				
2					
3	Customer	venue			
4	CALTRA Corp.	72K			
5	SANFRA Corp.	49K			
6	CATYOF Corp.	37K			
7	LOSVUG Corp.	26K			
8	ANIVUS Corp.	21K			
9	Top 5 Total	205K			
10					

Listing 11.10 Code That Creates a New Workbook per Region

```
Sub Top5ByRegionReport()
    ' Listing 11.10
    ' Produce a report of top 5 customers for each region
    Dim WSD As Worksheet
    Dim WSR As Worksheet
    Dim WBN As Workbook
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long

    Set WSD = Worksheets("PivotTable")

    ' Delete any prior pivot tables
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT
    WSD.Range("R1:AZ1").EntireColumn.Clear

    ' Define input area and set up a Pivot Cache
    FinalRow = WSD.Cells(Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Columns.Count). _
        End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:= _
        xlDatabase, SourceData:=PRange.Address)

    ' Create the Pivot Table from the Pivot Cache
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD. _
        Cells(2, FinalCol + 2), TableName:="PivotTable1")
```

continues

Listing 11.10 Continued

```
' Turn off updating while building the table
PT.ManualUpdate = True

' Set up the row fields
PT.AddFields RowFields:="Customer", ColumnFields:="Data", _
    PageFields:="Region"

' Set up the data fields
With PT.PivotFields("Revenue")
    .Orientation = xlDataField
    .Function = xlSum
    .Position = 1
    .NumberFormat = "#,##0,K"
    .Name = "Total Revenue"
End With

' Sort customers descending by sum of revenue
PT.PivotFields("Customer").AutoSort Order:=xlDescending, _
    Field:="Total Revenue"

' Show only the top 5 stores
PT.PivotFields("Customer").AutoShow Type:=xlAutomatic, Range:=xlTop, _
    Count:=5, Field:="Total Revenue"

' Ensure that we get zeros instead of blanks in the data area
PT.NullString = "0"

' Calc the pivot table
PT.ManualUpdate = False
PT.ManualUpdate = True
Ctr = 0

' Loop through each region
For Each PivItem In PT.PivotFields("Region").PivotItems
    Ctr = Ctr + 1
    PT.PivotFields("Region").CurrentPage = PivItem.Name
    PT.ManualUpdate = False
    PT.ManualUpdate = True

    ' Create a new blank workbook with one worksheet
    Set WBN = Workbooks.Add(xlWBATWorksheet)
    Set WSR = WBN.Worksheets(1)
    WSR.Name = PivItem.Name
    ' Set up Title for Report
    With WSR.[A1]
        .Value = "Top 5 Customers in the " & PivItem.Name & " Region"
        .Font.Size = 14
    End With

    ' Copy the pivot table data to row 3 of the report sheet
    ' Use offset to eliminate the page & title rows of the pivot table
    PT.TableRange2.Offset(3, 0).Copy
    WSR.[A3].PasteSpecial Paste:=xlPasteValuesAndNumberFormats
    LastRow = WSR.Cells(65536, 1).End(xlUp).Row
```

```
        WSR.Cells(LastRow, 1).Value = "Top 5 Total"

        ' Do some basic formatting
        ' Autofit columns, bold the headings, right-align
        WSR.Range(WSR.Range("A2"), WSR.Cells(LastRow, 8)).Columns.AutoFit
        Range("A3").EntireRow.Font.Bold = True
        Range("A3").EntireRow.HorizontalAlignment = xlRight
        Range("A3").HorizontalAlignment = xlLeft
        Range("B3").Value = "Revenue"

        Range("A2").Select

    Next PivItem

    ' Clear the pivot table
    PT.TableRange2.Clear
    Set PTCache = Nothing

    MsgBox Ctr & " Region reports have been created"

End Sub
```

Manually Filtering Two or More Items in a PivotField

In addition to setting up a calculated pivot item to display the total of a couple of products that make up a dimension, you can manually filter a particular PivotField.

For example, you have one client who sells shoes. In the report showing sales of sandals, he wants to see just the stores that are in warm-weather states. The code to hide a particular store is

```
PT.PivotFields("Store").PivotItems("Minneapolis").Visible = False
```

You need to be very careful never to set all items to `False`; otherwise, the macro ends with an error. This tends to happen more than you would expect. An application may first show products A and B and then on the next loop show products C and D. If you attempt to make A and B not visible before making C and D visible, no products will be visible along the `PivotField`, which causes an error. To correct this, always loop through all `PivotItems`, making sure to turn them back to visible before the second pass through the loop.

This process is easy in VBA. After building the table with `Product` in the page field, loop through to change the `Visible` property to show only the total of certain products:

```
    ' Make sure all PivotItems along line are visible
    For Each PivItem In _
        PT.PivotFields("Product").PivotItems
        PivItem.Visible = True
    Next PivItem

    ' Now - loop through and keep only certain items visible
    For Each PivItem In _
        PT.PivotFields("Product").PivotItems
        Select Case PivItem.Name
            Case "Landscaping/Grounds Care", _
                "Green Plants and Foliage Care"
```

```
                PivItem.Visible = True
            Case Else
                PivItem.Visible = False
        End Select
    Next PivItem
```

Controlling the Sort Order Manually

If your company has been reporting regions in the sequence South, North, West forever, it is an uphill battle getting managers to accept seeing the report ordered North, South, West just because this is the default alphabetical order offered by pivot tables.

Strangely enough, Microsoft offers a bizarre method for handling a custom sort order in a pivot table. It's called a *manual sort order*. To change the sort order in the user interface, you simply go to a cell in the pivot table that contains North, type the word **`South`**, and press Enter. As if by magic, North and South switch places. Of course, all the numbers for North move to the appropriate column.

The VBA code to do a manual sort involves setting the `Position` property for a specific `PivotItem`. This is somewhat dangerous because you don't know whether the underlying fields will have data for South on any given day. Be sure to set error checking to resume in case South doesn't exist today:


```
On Error Resume Next
PT.PivotFields("Region").PivotItems("South").Position = 1
On Error GoTo 0
```

Using Sum, Average, Count, Min, Max, and More

So far, every example in this chapter has involved summing data. It is also possible to get an average, minimum, or maximum of data. In VBA, change the `Function` property of the data field and give the data field a unique name. For example, the following code fragment produces five different summaries of the revenue field, each with a unique name:

```
' Set up the data fields
With PT.PivotFields("Revenue")
    .Orientation = xlDataField
    .Function = xlSum
    .Position = 1
    .NumberFormat = "#,##0,K"
    .Name = "Total Revenue"
End With

With PT.PivotFields("Revenue")
    .Orientation = xlDataField
    .Function = xlCount
    .Position = 2
    .NumberFormat = "#,##0"
    .Name = "Number Orders"
End With

With PT.PivotFields("Revenue")
    .Orientation = xlDataField
```

```
    .Function = xlAverage
    .Position = 3
    .NumberFormat = "#,##0"
    .Name = "Average Revenue"
End With

With PT.PivotFields("Revenue")
    .Orientation = xlDataField
    .Function = xlMin
    .Position = 4
    .NumberFormat = "#,##0"
    .Name = "Smallest Order"
End With

With PT.PivotFields("Revenue")
    .Orientation = xlDataField
    .Function = xlMax
    .Position = 5
    .NumberFormat = "#,##0"
    .Name = "Largest Order"
End With
```

The resultant pivot table provides a number of statistics about the average revenue, largest order, smallest order, and so on.

Creating Report Percentages

In addition to the available choices, such as Sum, Min, Max, and Average, you can use another set of pivot table options called the *calculation options*. They allow you to show a particular field as a percentage of the total, a percentage of the row, a percentage of the column, or as the percent difference from the previous or next item. All these settings are controlled through the `.Calculation` property of the page field.

The valid properties for `.Calculation` are `xlPercentOf`, `xlPercentOfColumn`, `xlPercentOfRow`, `xlPercentOfTotal`, `xlRunningTotal`, `xlPercentDifferenceFrom`, `xlDifferenceFrom`, `xlIndex`, and `xlNoAdditionalCalculation`. Each has its own unique set of rules. Some require that you specify a `BaseField`, and others require that you specify both a `BaseField` and `BaseItem`. The following sections provide some specific examples.

Percentage of Total

To get the percentage of the total, specify `xlPercentOfTotal` as the `.Calculation` property for the page field:

```
' Set up a percentage of total
With PT.PivotFields("Revenue")
    .Orientation = xlDataField
    .Caption = "PctOfTotal"
    .Function = xlSum
    .Position = 2
    .NumberFormat = "#0.0%"
    .Calculation = xlPercentOfTotal
End With
```

Percentage Growth from Previous Month

With ship months going down the columns, you might want to see the percentage of revenue growth from month to month. You can set up this arrangement with the `xlPercentDifferenceFrom` setting. In this case, you must specify that the `BaseField` is `"Invoice Date"` and that the `BaseItem` is something called `"(previous)"`:

```
' Set up % change from prior month
With PT.PivotFields("Revenue")
    .Orientation = xlDataField
    .Function = xlSum
    .Caption = "%Change"
    .Calculation = xlPercentDifferenceFrom
    .BaseField = "Invoice Date"
    .BaseItem = "(previous)"
    .Position = 3
    .NumberFormat = "#0.0%"
End With
```

Note that with positional calculations, you cannot us the AutoShow or AutoSort method. This is too bad; it would be interesting to sort the customers high to low and to see their sizes in relation to each other.

Percentage of a Specific Item

You can use the `xlPercentDifferenceFrom` setting to express revenues as a percentage of the California market sales:

```
' Show revenue as a percentage of California
With PT.PivotFields("Revenue")
    .Orientation = xlDataField
    .Function = xlSum
    .Caption = "% of California"
    .Calculation = xlPercentDifferenceFrom
    .BaseField = "Market"
    .BaseItem = "California"
    .Position = 3
    .NumberFormat = "#0.0%"
End With
```

Running Total

Setting up a running total is not intuitive; to do this, you must define a `BaseField`. In this example, Invoice Date runs down the column. To define a running total column for revenue, you must specify that `BaseField` is `"Invoice Date"`:

```
' Set up Running Total
With PT.PivotFields("Revenue")
    .Orientation = xlDataField
    .Function = xlSum
    .Caption = "YTD Total"
    .Calculation = xlRunningTotal
    .Position = 4
    .NumberFormat = "#,##0,K"
    .BaseField = "Invoice Date"
End With
```

Figure 11.22 shows the results of a pivot table with three custom calculation settings, as discussed earlier.

Figure 11.22
This pivot table presents four views of Sum of Revenue. Column T is the normal calculation. Column U is % of Total. Column V is % change from previous month. Column W is the running total.

	R	S	T	U	V	W
1						
2			Data			
3	Years	InvoiceDate	Sum of Revenue	PctOfTotal	%Change	YTD Total
4	⊟2006	Jan	197K	2.1%		197K
5		Feb	313K	3.4%	58.7%	510K
6		Mar	200K	2.2%	-36.1%	710K
7		Apr	269K	2.9%	34.8%	979K
8		May	278K	3.0%	3.0%	1,257K
9		Jun	252K	2.7%	-9.3%	1,509K
10		Jul	276K	3.0%	9.6%	1,784K
11		Aug	275K	3.0%	-0.4%	2,059K
12		Sep	237K	2.6%	-13.8%	2,296K
13		Oct	284K	3.1%	20.0%	2,580K
14		Nov	266K	2.9%	-6.3%	2,847K
15		Dec	213K	2.3%	-20.1%	3,060K

Using New Pivot Table Features in Excel 2007

Pivot tables offer a variety of new features in Excel 2007. The new label and value filters, conditional formatting, table formatting, and layout views are significant improvements to the pivot table environment.

If you want to utilize any of these features, the pivot table must exist in a file that is stored in Excel 2007 file format. If your file is in compatibility mode, none of the new features are available in the user interface nor in VBA.

Similarly, if you use any of these features, the code runs only in Excel 2007. There is no hope of going backward to share the code with someone using Excel 2003.

Using the New Filters

In previous versions of Excel, the filtering feature allowed you to choose one or more pivot items from a drop-down list. The only conceptual filter was the top 10 AutoShow filter.

Excel 2007 offers new conceptual filters that are easy to access. In the PivotTable Field List, hover the mouse cursor over any active field in the field list portion of the dialog box. In the drop-down that appears, you can choose either Label Filters, Date Filters, or Value Filters.

In Figure 11.23, the flyout menu shows the list of Label filters available for the Branch field.

To apply a label filter in VBA, use the `PivotFilters.Add` method. The following code filters to the Branches that start with 1:

```
PT.PivotFields("Branch").PivotFilters.Add _
    Type:=xlCaptionBeginsWith, Value1:="1"
```

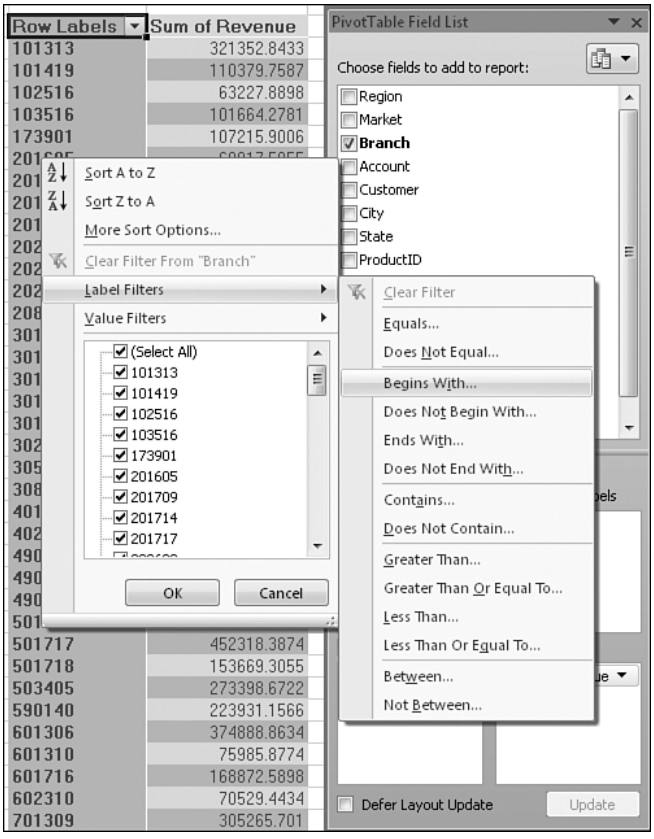


Figure 11.23
You can easily choose all the Branch items that meet your criteria.

To clear the filter from the Branch field, use the `ClearAllFilters` method:

```
PT.PivotFields("Branch").ClearAllFilters
```

To apply a date filter to the invoice date field to find records from this week, use this code:

```
PT.PivotFields("InvoiceDate").PivotFilters.Add Type:=xlThisWeek
```

The value filters allow you to filter one field based on the value of another field. For example, to find all the branches where the total revenue is over $100,000, you would use this code:

```
PT.PivotFields("Branch").PivotFilters.Add _
    Type:=xlValueIsGreaterThan, _
    DataField:=PT.PivotFields("Sum of Revenue"), _
    Value1:=100000
```

Other value filters might allow you to specify that you want branches where the revenue is between $50,000 and $100,000. In this case, you would specify one limit as `Value1` and the second limit as `Value2`:

```
PT.PivotFields("Branch").PivotFilters.Add _
    Type:=xlValueIsBetween, _
    DataField:=PT.PivotFields("Sum of Revenue"), _
    Value1:=50000, Value2:=100000
```

Table 11.3 lists all the possible filter types.

Table 11.3 Filter Types

Filter Type	Description
xlBefore	Filters for all dates before a specified date
xlBeforeOrEqualTo	Filters for all dates on or before a specified date
xlAfter	Filters for all dates after a specified date
xlAfterOrEqualTo	Filters for all dates on or after a specified date
xlAllDatesInPeriodJanuary	Filters for all dates in January
xlAllDatesInPeriodFebruary	Filters for all dates in February
xlAllDatesInPeriodMarch	Filters for all dates in March
xlAllDatesInPeriodApril	Filters for all dates in April
xlAllDatesInPeriodMay	Filters for all dates in May
xlAllDatesInPeriodJune	Filters for all dates in June
xlAllDatesInPeriodJuly	Filters for all dates in July
xlAllDatesInPeriodAugust	Filters for all dates in August
xlAllDatesInPeriodSeptember	Filters for all dates in September
xlAllDatesInPeriodOctober	Filters for all dates in October
xlAllDatesInPeriodNovember	Filters for all dates in November
xlAllDatesInPeriodDecember	Filters for all dates in December
xlAllDatesInPeriodQuarter1	Filters for all dates in Quarter 1
xlAllDatesInPeriodQuarter2	Filters for all dates in Quarter 2
xlAllDatesInPeriodQuarter3	Filters for all dates in Quarter 3
xlAllDatesInPeriodQuarter4	Filters for all dates in Quarter 4
xlBottomCount	Filters for the specified number of values from the bottom of a list
xlBottomPercent	Filters for the specified percentage of values from the bottom of a list
xlBottomSum	Sums the values from the bottom of the list
xlCaptionBeginsWith	Filters for all captions beginning with the specified string

continues

Table 11.3 Continued

Filter Type	Description
xlCaptionContains	Filters for all captions that contain the specified string
xlCaptionDoesNotBeginWith	Filters for all captions that do not begin with the specified string
xlCaptionDoesNotContain	Filters for all captions that do not contain the specified string
xlCaptionDoesNotEndWith	Filters for all captions that do not end with the specified string
xlCaptionDoesNotEqual	Filters for all captions that do not match the specified string
xlCaptionEndsWith	Filters for all captions that end with the specified string
xlCaptionEquals	Filters for all captions that match the specified string
xlCaptionIsBetween	Filters for all captions that are between a specified range of values
xlCaptionIsGreaterThan	Filters for all captions that are greater than the specified value
xlCaptionIsGreaterThanOrEqualTo	Filters for all captions that are greater than or match the specified value
xlCaptionIsLessThan	Filters for all captions that are less than the specified value
xlCaptionIsLessThanOrEqualTo	Filters for all captions that are less than or match the specified value
xlCaptionIsNotBetween	Filters for all captions that are not between a specified range of values
xlDateBetween	Filters for all dates that are between a specified range of dates
xlDateLastMonth	Filters for all dates that apply to the previous month
xlDateLastQuarter	Filters for all dates that apply to the previous quarter
xlDateLastWeek	Filters for all dates that apply to the previous week
xlDateLastYear	Filters for all dates that apply to the previous year

Filter Type	Description
xlDateNextMonth	Filters for all dates that apply to the next month
xlDateNextQuarter	Filters for all dates that apply to the next quarter
xlDateNextWeek	Filters for all dates that apply to the next week
xlDateNextYear	Filters for all dates that apply to the next year
xlDateThisMonth	Filters for all dates that apply to the current month
xlDateThisQuarter	Filters for all dates that apply to the current quarter
xlDateThisWeek	Filters for all dates that apply to the current week
xlDateThisYear	Filters for all dates that apply to the current year
xlDateToday	Filters for all dates that apply to the current date
xlDateTomorrow	Filters for all dates that apply to the next day
xlDateYesterday	Filters for all dates that apply to the previous day
xlNotSpecificDate	Filters for all dates that do not match a specified date
xlSpecificDate	Filters for all dates that match a specified date
xlTopCount	Filters for the specified number of values from the top of a list
xlTopPercent	Filters for the specified percentage of values from a list
xlTopSum	Sums the values from the top of the list
xlValueDoesNotEqual	Filters for all values that do not match the specified value
xlValueEquals	Filters for all values that match the specified value
xlValueIsBetween	Filters for all values that are between a specified range of values
xlValueIsGreaterThan	Filters for all values that are greater than the specified value
xlValueIsGreaterThanOrEqualTo	Filters for all values that are greater than or match the specified value
xlValueIsLessThan	Filters for all values that are less than the specified value
xlValueIsLessThanOrEqualTo	Filters for all values that are less than or match the specified value

continues

Table 11.3 Continued

Filter Type	Description
`xlValueIsNotBetween`	Filters for all values that are not between a specified range of values
`xlYearToDate`	Filters for all values that are within one year of a specified date

Applying a Table Style

The Design ribbon offers two groups dedicated to formatting the pivot table, as shown in Figure 11.24. The PivotTable Style Options group has four check boxes that modify the styles in the PivotTable Styles Gallery.

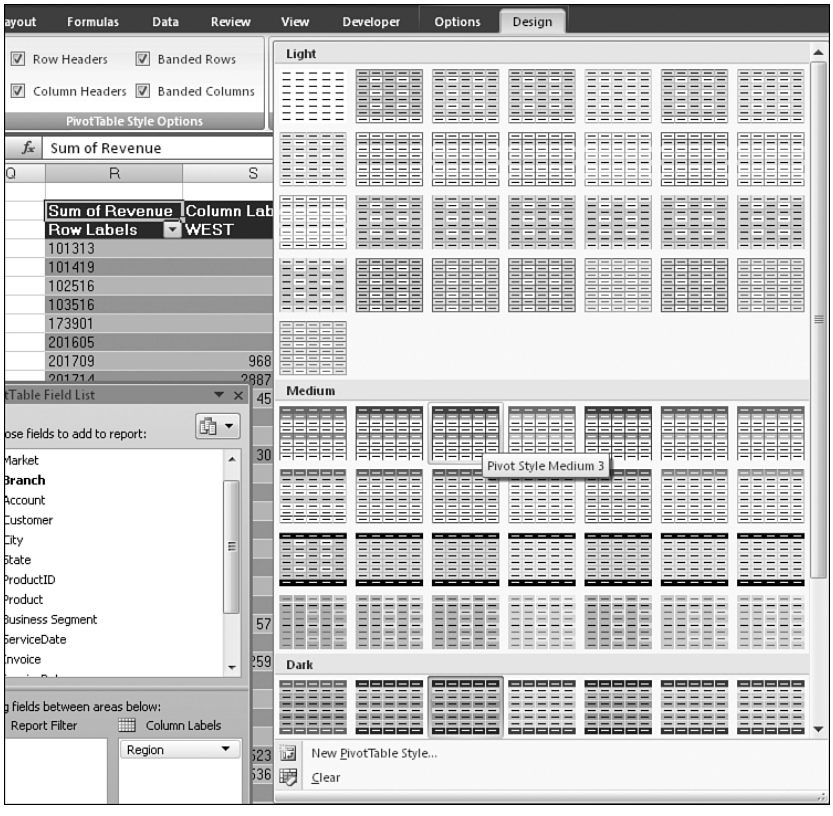


Figure 11.24 The four check boxes and gallery of styles offer many variations for formatting the pivot table.

The following four lines of code are equivalent to turning on all four settings in the PivotTable Style Options group:

```
PT.ShowTableStyleRowHeaders = True
PT.ShowTableStyleColumnHeaders = True
PT.ShowTableStyleRowStripes = True
PT.ShowTableStyleColumnStripes = True
```

To apply a table style from the gallery, use the `TableStyle2` property. If you want to get the correct name, it might be best to record a macro:

```
' Format the pivot table
PT.ShowTableStyleRowStripes = True
PT.TableStyle2 = "PivotStyleMedium3"
```

In Figure 11.24, the ToolTip shows a style named Pivot Style Medium 3.

> **NOTE**
>
> Previous versions of Excel offered an AutoFormat feature for pivot tables. This feature was annoying because it actually changed the layout of your pivot table. That obsolete command used the `TableStyle` property; hence, Excel 2007 had to use `TableStyle2` as the property name for the new style tables.

> **CAUTION**
>
> It is possible to create custom table styles. If you have a custom table style named `MyStyle44` and use this name in a macro, the macro will run fine on your computer but may not run on anyone else's computer. To alleviate the chance of a runtime error, you use `On Error Resume Next` before applying `TableStyle2`.

Changing the Layout

The Layout group of the Design ribbon contains four drop-downs. These drop-downs control the location of subtotals (top or bottom), the presence of grand totals, the report layout, and the presence of blank rows.

Subtotals can appear either at the top or bottom of a group of pivot items. The `SubtotalLocation` property applies to the entire pivot table; valid values are `xlAtBottom` or `xlAtTop`:

```
PT.SubtotalLocation:=xlAtTop
```

Grand totals can be turned on or off for rows or columns. The following code turns them off for both:

```
PT.ColumnGrand = False
PT.RowGrand = False
```

There are three settings for the report layout. The Tabular layout is similar to the default layout in Excel 2003. The Outline layout was optionally available in Excel 2003. The Compact layout is new in Excel 2007.

Excel can remember the last layout used and will apply it to additional pivot tables created in the same Excel session. For this reason, you should always explicitly choose the layout that you want. Use the `RowAxisLayout` method; valid values are `xlTabularRow`, `xlOutlineRow`, or `xlCompactRow`:

```
PT.RowAxisLayout xlTabularRow
PT.RowAxisLayout xlOutlineRow
PT.RowAxisLayout = xlCompactRow
```

In Excel 2007, you can add a blank line to the layout after each group of pivot items. Although the Design ribbon offers a single setting to affect the entire pivot table, the setting is actually applied to each individual pivot field individually. The macro recorder responds by recording a dozen lines of code for a pivot table with 12 fields. You can intelligently add a single line of code for the outer row field(s):

```
PT.PivotFields("Region").LayoutBlankLine = True
```

Applying a Data Visualization

Excel 2007 offers fantastic new data visualizations such as icon sets, color gradients, and in-cell data bars. When you apply a visualization to a pivot table, you should exclude the total rows from the visualization.

If you have 30 branches that average $50,000 in revenue each, the total for the 30 branches is $1.5 million. If you include the total in the data visualization, the total gets the largest bar, and all the branch records have tiny bars.

In the Excel user interface, you always want to use the Add Rule or Edit Rule choice to choose the option All Cells Showing "Sum of Revenue" for "Branch," as shown in Figure 11.25.

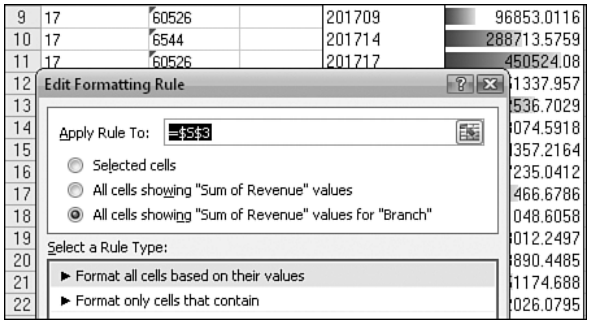

Edit Formatting Rule
Apply Rule To:
=$5$3
Selected cells
All cells showing "Sum of Revenue" values
All cells showing "Sum of Revenue" values for "Branch"
Select a Rule Type:
► Format all cells based on their values
► Format only cells that contain

Figure 11.25 To create meaningful visualizations in your pivot table, exclude the totals by choosing the third option at the top of this dialog box.

The code in Listing 11.11 adds a pivot table and applies a data bar to the revenue field.

Listing 11.11 Code That Creates a Pivot Table with Data Bars

```
Sub CreatePivotDataBar()
    ' Listing 11.11
    Dim WSD As Worksheet
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long
    Set WSD = Worksheets("PivotTable")
```

```
    ' Delete any prior pivot tables
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT
    WSD.Range("R1:AZ1").EntireColumn.Clear

    ' Define input area and set up a Pivot Cache
    FinalRow = WSD.Cells(Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Columns.Count). _
        End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)
    Set PTCache = ActiveWorkbook.PivotCaches.Add(SourceType:= _
        xlDatabase, SourceData:=PRange.Address)

    ' Create the Pivot Table from the Pivot Cache
    Set PT = PTCache.CreatePivotTable(TableDestination:=WSD. _
        Cells(2, FinalCol + 2), TableName:="PivotTable1")

    ' Turn off updating while building the table
    PT.ManualUpdate = True

    ' Set up the row & column fields
    PT.AddFields RowFields:="Branch", _
        ColumnFields:="Data"

    ' Set up the data fields
    With PT.PivotFields("Revenue")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
    End With

    ' Calc the pivot table
    PT.ManualUpdate = False
    PT.ManualUpdate = True

    ' Apply a Databar
    PT.TableRange2.Cells(3, 2).Select
    Selection.FormatConditions.AddDatabar
    Selection.FormatConditions(1).ShowValue = True
    Selection.FormatConditions(1).SetFirstPriority
    With Selection.FormatConditions(1)
        .MinPoint.Modify newtype:=xlConditionValueLowestValue
        .MaxPoint.Modify newtype:=xlConditionValueHighestValue
    End With
    With Selection.FormatConditions(1).BarColor
        .ThemeColor = xlThemeColorAccent3
        .TintAndShade = -0.499984740745262
    End With
    Selection.FormatConditions(1).ScopeType = xlFieldsScope

    WSD.Activate
    Range("R1").Select

End Sub
```

Understanding Special Considerations for Excel 97

Pivot tables and VBA took a radical turn in Excel 2000. In Excel 2000, Microsoft introduced the `PivotCache` object. This object allows you to define one pivot cache and then build many pivot reports from the pivot cache.

Officially, Microsoft quit supporting Excel 97 a few years ago. But, in practical terms, many companies are still using Excel 97. If you need your code to work on a legacy platform, you should be aware of the way pivot tables were created in Excel 97.

In Excel 97, you used the `PivotTableWizard` method. Take a look at the code for building a simple pivot table showing revenue by region and product. Where current code uses two steps (add `PivotCache` and then use `CreatePivotTable`), Excel 97 used just one step, using the `PivotTableWizard` method to create the table:

```
Sub PivotExcel97Compatible()
    ' Pivot Table Code for Excel 97 Users
    Dim WSD As Worksheet
    Dim PT As PivotTable
    Dim PRange As Range
    Dim FinalRow As Long

    Set WSD = Worksheets("PivotTable")

    ' Delete any prior pivot tables
    For Each PT In WSD.PivotTables
        PT.TableRange2.Clear
    Next PT

    ' Define input area
    FinalRow = WSD.Cells(Rows.Count, 1).End(xlUp).Row
    FinalCol = WSD.Cells(1, Columns.Count). _
        End(xlToLeft).Column
    Set PRange = WSD.Cells(1, 1).Resize(FinalRow, FinalCol)

    ' Create pivot table using PivotTableWizard
    Set PT = WSD.PivotTableWizard(SourceType:=xlDatabase, _
        SourceData:=PRange.Address, _
        TableDestination:="R2C18", TableName:="PivotTable1")

    PT.ManualUpdate = True
    ' Set up the row fields
    PT.AddFields RowFields:="Region", ColumnFields:="Product"

    ' Set up the data fields
    With PT.PivotFields("Revenue")
        .Orientation = xlDataField
        .Function = xlSum
        .Position = 1
        .NumberFormat = "#,##0,K"
        .Name = "Total Revenue"
    End With

    PT.ManualUpdate = False
    PT.ManualUpdate = True
End Sub
```

Next Steps

In the next chapter, you learn a myriad of techniques for handling common questions and issues with pivot tables.